Progress®



Manage ABL Applications

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....11
Purpose....11
Audience....11
Organization....12
Using ABL documentation....13
References to ABL compiler and run-time features....13
References to ABL data types....13
Typographical conventions....14
Examples of syntax descriptions....15
Long syntax descriptions split across lines....16
Complex syntax descriptions with both required and optional elements....17
OpenEdge messages....17
Obtaining more information about OpenEdge messages....18

Part I: ABL and R-code Deployment and Management....19

Chapter 1: ABL Client Deployment Overview....21
Client deployment process....21
Client deployment and administration tasks....22
Additional deployment documentation....22

Chapter 2: Managing Client Access to Databases....25
Connecting clients to databases....25
Logical database names....26
Connection modes....26
Connection parameters....27
Connection techniques....27
Connecting to a non-OpenEdge database....33
Connection denials when the user count is exceeded....34
Reducing connection times....34
Starting clients without connecting to a database....35
Selecting a working database for client use....35
Selecting a database with the Data Dictionary....35
Selecting a working database with the Data Dictionary when using a character interface....36
Selecting a database with the Data Administration tool....36
Disconnecting databases for clients....36
Using the Data Dictionary to disconnect....36

Disconnecting a database with the Data Dictionary when using a character interface....36
Using the Data Administration tool to disconnect....37

Chapter 3: Maintaining Application Security....39

Database table- and field-level security....40
Specifying run-time permissions checking....41
Setting table and field permissions....42
Determining the privileges of the blank user ID....45
Compile-time security....46
Run-time security....47
Application support....47
OpenEdge support....47
Operating system security....47
Designating a security administrator....48
ABL security administration....49
Auditing administration....49
Transparent data encryption administration....49

Chapter 4: Maintaining User Environments....51

Maintaining the Windows user environment....51
Using the INI2REG utility....52
Windows icons for running OpenEdge....79
Starting the OpenEdge client....79
Modifying client icons....80
Windows manifest file....80
Environments with OpenEdge GUI for .NET applications....81
Maintaining the UNIX user environment....82
Maintaining the buildenv script....83
Maintaining the PROTERMCAP file....83
PROTERMCAP syntax....84
Terminal capabilities entries....87
Vermont Views key function capabilities....91
Pointer to key functions....93
Specifying colors....93
Specifying the cursor motion capability....94
Specifying keyboard mappings....96
Extended character support entries....101
Creating a PROTERMCAP entry....102
Testing terminal entries....102
Setting up the terminal....102
Setting the terminal type....102
Setting the PROTERMCAP environment variable....103
Reverting to the default PROTERMCAP file....103

Copying an existing terminal entry....104

Chapter 5: Managing Client Performance....105

Procedure loading and execution....105
Using procedure libraries to improve r-code performance....106
R-code execution environment....106
Monitoring r-code activity....108
Distributing OpenEdge files on a network....111
Distributing OpenEdge system files....112
Distributing application files....112
Distributing temporary files....112
Distributing log files....113
Temporary file I/O....113
Sorting I/O....114

Chapter 6: Managing Procedure Libraries....115

Library overview....116
Loading r-code from a file....117
Loading r-code from a standard library....117
Loading r-code from a memory-mapped library....118
Setting up a library....118
Running members from a library....119
Using an absolute pathname....119
Using a relative pathname....119
Related functions....120
Libraries and PROPATH....120
Memory and network considerations....121
Using the PROLIB utility....122
Using wild cards....123
Creating a standard library....124
Generating a memory-mapped library....124
Adding files to a standard library....125
Replacing files in a standard library....126
Deleting files from a standard library....126
Listing the contents of a library....127
Extracting files from a standard library....128
Extracting files to your current directory....129
Compressing a standard library....130
PROLIB command examples....130
Code page compatibility....131

Chapter 7: Managing Print Devices....133

Printing in OpenEdge....133

OUTPUT TO statement....134
OUTPUT THROUGH statement....135
Printing in Windows....135
Setting up output-routing....136
Using a startup procedure....136
Using an output-routing table....137
Using interactive output-routing....138
Printing on UNIX....139
Configuring a printer....139
Creating a printer-control table....140

Part II: Deployment Considerations....143

Chapter 8: Choosing a Code Format....145
Unencrypted source....145
Development product requirements for unencrypted source....145
User product requirements for unencrypted source....146
Advantages of unencrypted source....146
Possible drawbacks of unencrypted source....146
Encrypted source....147
Development product requirements for encrypted source....147
User product requirements for encrypted source....147
Advantages of encrypted source....147
Possible drawbacks of encrypted source....148
R-code....148
Developer product requirements and r-code portability....148
User product requirements....148
Advantages of r-code....148
Possible drawbacks of r-code....149

Chapter 9: Initial Deployment....151
Deploying database structures and application files....152
Unencrypted source....153
Encrypted source....154
CRC r-code....157
Deploying OpenEdge GUI for .NET applications....159
Installing .NET Framework....159
Creating an assemblies directory....159
Deploying application source code....160

Chapter 10: Upgrades....161
Deploying upgrades....162

Modifying the application procedures....163
Unencrypted source....163
Encrypted source files....163
CRC-based r-code....163
Modifying the application database....163
Upgrading with updated .df and encrypted source....164
Upgrading with updated .df and CRC-based r-code....166
OpenEdge product upgrades....167

Chapter 11: Deployment Topics and Tasks....169
Encrypting source code procedures....169
XCODE utility....169
Preparing to use XCODE....170
Compilation....171
Using the BUNDLE utility....172
The bundle.c program....172
The unbundle.c program....173
Examples using BUNDLE....174
Dumping and loading....177
MKDUMP utility....177
Using your dump/reload facility....178
Restricting database access....179

Appendix A: Building OpenEdge Executables....181
Requirements....182
Building executables on UNIX....182
Preparing the UNIX software environment....182
Building a customized executable on UNIX....183
Using the customized UNIX executable....184
Building and running the HLC examples for UNIX....185
Building executables in Windows....185
Preparing the Windows software environment....185
Building a customized executable in Windows....186
Troubleshooting....187

Appendix B: R-code Features and Functions....189
R-code structure....190
Factors that affect r-code size....191
R-code file segment layout....191
Procedure libraries....192
Procedure Libraries and PROPATH....193
R-code execution....193
Standard r-code execution environment....194

Memory-mapped r-code execution environment 196
R-code directory management 198
R-code execution environment statistics 198
R-code portability 201
Display architectures....201
Database types....201
R-code version....202
R-code portability and operating system-specific statements....202
Byte order of integers in R-code....202
Examples of portable and nonportable r-code....202
Code page compatibility....203
Database CRCs and time stamps 203
CRC validation 203
CRC calculation....204
CRC-based deployment 206
R-code access authorization 207
R-code CRCs and procedure integrity 207
Assigning CRCs to schema triggers 208
Validating CRCs for schema triggers 208
RCODE-INFO handle 208

Appendix C: OpenEdge Application Limits....213
Input/output limits....213
Sorting limits....214
Name limits....214
Compiler limits....215

Appendix D: Deployment Utilities and Scripts....217
Templates 217
Utilities summary....218
DBRSTRCT utility 218
MKDUMP utility 219
XCODE utility 220
_tlr Module 221

Index....223

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

Purpose

This book describes how to manage ABL (Advanced Business Language) client applications. The first part describes the ABL client application deployment process and application administration concepts and procedures. The second part of this book describes the steps required to deploy ABL applications and the trade-offs between different deployment scenarios. It also describes components provided with OpenEdge® that help you through the deployment process. The final part contains appendices which supplement the rest of the material.

Audience

The first part of this book is intended for OpenEdge application administrators. The second part is intended for experienced ABL programmers preparing applications for wider distribution or resale.

Organization

ABL and R-code Deployment and Management

ABL Client Deployment Overview on page 21

Presents an overview of the client deployment process, and outlines the deployment and administration tasks you must perform.

Managing Client Access to Databases on page 25

Describes how to connect and disconnect databases for OpenEdge clients.

Maintaining Application Security on page 39

Describes how OpenEdge implements application security, including designating security administrators and setting table and field permissions.

Maintaining User Environments on page 51

Describes how to maintain OpenEdge user interface environments in Windows and on UNIX.

Managing Client Performance on page 105

Describes how to monitor and tune OpenEdge client application performance.

Managing Procedure Libraries on page 115

Describes ABL procedure libraries and how to use the PROLIB utility.

Managing Print Devices on page 133

Describes how to set up print devices on OpenEdge-supported operating systems.

Deployment Considerations

Choosing a Code Format on page 145

Describes the different types of code formats you can choose to deploy an OpenEdge application.

Initial Deployment on page 151

Describes the initial deployment process using the different code formats.

Upgrades on page 161

Describes how to upgrade OpenEdge applications.

Deployment Topics and Tasks on page 169

Describes how to perform several tasks related to deploying OpenEdge applications.

Appendices

Building OpenEdge Executables on page 181

Describes the process and requirements for building customized OpenEdge executables. It also provides instructions for using the Make utility in Windows and on UNIX.

R-code Features and Functions on page 189

Describes the structure and management of ABL r-code, the executable code into which OpenEdge compiles ABL procedures. It also describes techniques for tuning r-code size and performance, and the use of cyclic redundancy checks (CRCs) to maintain r-code and database integrity.

OpenEdge Application Limits on page 213

Describes the OpenEdge limits you must consider when developing an OpenEdge client application.

Deployment Utilities and Scripts on page 217

Provides reference information about utilities and scripts you can use when deploying your application.

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all `UPPERCASE`, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever *character* appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever *decimal* appears, this is a reference to the `DECIMAL` data type.
- Wherever *numeric* appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.

Convention	Description
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, ACCUM is a keyword, and *aggregate* and *expression* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } . . .
       | { record [ EXCEPT field . . . ] }
```

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

```
OpenEdge-install-dir/bin/pro
```

2. Press **F3** to access the menu bar, then choose **Help** > **Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose **File** > **Exit**.

I

ABL and R-code Deployment and Management

For details, see the following topics:

- ABL Client Deployment Overview
- Managing Client Access to Databases
- Maintaining Application Security
- Maintaining User Environments
- Managing Client Performance
- Managing Procedure Libraries
- Managing Print Devices

1

ABL Client Deployment Overview

Deploying an ABL (Advanced Business Language) client application is a process that involves developing an application and installing that application in end-user environments. Once you have deployed your application, you must manage the end-user environments and system resources for your users.

This chapter presents an overview of the client deployment process, and outlines the deployment and administration tasks you must perform.

For details, see the following topics:

- Client deployment process
- Client deployment and administration tasks
- Additional deployment documentation

Client deployment process

While deployment requirements vary between applications, you follow a basic deployment process to deploy any ABL client application.

The client deployment process consists of three basic phases:

- **Planning** — The phase in which you identify your deployment requirements and define your deployment strategy
- **Development** — The phase in which you design, develop, and debug your client application
- **Installation and Configuration** — The phase in which you deliver your client application to end-users, and set up the end-user environment

Each phase of the client deployment process has its own set of requirements and dependencies that you must consider when defining your client deployment strategy. Decisions made in an earlier phase will affect the following phases and the process as a whole.

Progress Software Corporation provides you with the following services to help you identify your client deployment options and accomplish the client deployment process for your application:

- The Progress Software Consulting Services organization which can assist you in all phases of the client deployment process
- The Progress Technical Services staff which can answer your questions about OpenEdge tools, ABL usage, and database administration

Client deployment and administration tasks

The following are some of the ABL client application deployment and administration tasks you must perform:

- Manage client access to databases
- Maintain application security
- Manage client performance
- Maintain user environments
- Manage procedure libraries
- Manage print devices

The remaining chapters in this guide provide detailed information about each of these tasks, as well as the procedures and utilities you use to accomplish them. The appendixes provide useful information about building OpenEdge executables and OpenEdge application limits.

Note: Certain deployment and administration tasks will differ among operating systems.

For information about designing and coding an ABL client application, see OpenEdge Getting Started: ABL Essentials.

For information about creating and maintaining an OpenEdge RDBMS, see OpenEdge Data Management: Database Administration.

For information about AppServer deployment and administration tasks, see OpenEdge Application Server: Developing AppServer Applications.

Additional deployment documentation

This manual provides information about deploying an OpenEdge application. For information about other types of deployment, see the manuals listed in the following table.

Table 1: Deployment information in OpenEdge manuals

Product area	Manual
AppServer	*OpenEdge Application Server: Developing AppServer Applications* *OpenEdge Application Server: Developing AppServer Applications*

Product area	Manual
Database	*Manage the OpenEdge Database*
Open Client	*Use the Open Client Toolkit* *Develop .NET Open Clients* *Develop Java Open Clients* *Develop Web Services for OpenEdge*
WebClient	*Develop Web Clients*
WebSpeed	*OpenEdge Getting Started: WebSpeed Essentials* *OpenEdge Application Server: Administration*
Web Services	*Develop Web Services for OpenEdge* *OpenEdge Application Server: Administration*
Progress Dynamics	*OpenEdge Development: Progress Dynamics Administration*
Internationalization	*Internationalize ABL Applications*

2

Managing Client Access to Databases

This chapter describes how to connect and disconnect databases for ABL (Advanced Business Language) clients and provides information about the following topics:

For details, see the following topics:

- Connecting clients to databases
- Selecting a working database for client use
- Disconnecting databases for clients

Connecting clients to databases

To access a database, you must connect to the database. You can connect to one or more databases during a single OpenEdge session. You can also start an OpenEdge session without connecting to a database, and connect later.

Note: On UNIX, the kernel configuration's file descriptor limit imposes a practical limit on connected databases. "Unable to open file" or "Unable to allocate I-Node" messages often mean your system's file descriptors have been exhausted. "Unable to allocate shared memory" often means your shared memory has been exhausted on your system. For more information about kernel configurations, see *Configure OpenEdge*.

When connecting to databases, consider the following factors:

- Logical database names on page 26
- Connection modes on page 26

- Connection parameters on page 27
- Connection techniques on page 27
- Connecting to a non-OpenEdge database on page 33
- Connection denials when the user count is exceeded on page 34
- Reducing connection times on page 34
- Starting clients without connecting to a database on page 35

Logical database names

A *logical database name* is a database reference that represents the name of a connected physical database. OpenEdge uses the logical database name to resolve database references. When a procedure is compiled against a database, OpenEdge stores the logical database name in the procedure's r-code. When a procedure executes, its database references must match the logical name of a connected database.

A database has a logical name whenever it is connected. When you connect to a database, OpenEdge automatically assigns a default logical name for the current session. The default logical name consists of the physical database name without the `.db` suffix. For example, if you connect to a database with the physical name `mydb1.db`, the default logical database name is `mydb1`. Alternatively, you can use the Logical Database Name (-ld) startup parameter to specify a logical database name other than the default.

For example, to assign the logical name `firstdb` to the physical database `mydb1.db` using the PRO command, enter the following command:

```
pro mydb1 -ld firstdb
```

Logical database names allow you to run procedures on different physical databases without recompiling. To run a compiled procedure on a second physical database, the database must meet the following criteria:

- It must have the identical structure (data definitions) as the database you compiled the procedure against.
- It must be connected with the same logical name as the database you compiled the procedure against.

A database connection fails if the logical database name of the database that you are connecting has the same logical name as an already connected database. If you try to connect a database with a logical database name that matches the logical database name of an existing connected database of the same database type (OpenEdge RDBMS, ORACLE, etc.), OpenEdge assumes that database is connected and ignores the request.

You can change a database's logical name in an ABL procedure with the `CREATE ALIAS` statement. For more information about logical and physical database names and aliases, and the `CREATE ALIAS` statement, see *ABL Reference* and *OpenEdge External Programming Interface Reference*.

Connection modes

A database is connected in one of the following connection modes:

- **Single-user** — Only the current OpenEdge session can access the specified database. If the database is already in use by another user, you cannot connect to the database from the current OpenEdge session.
- **Multi-user client** — The current OpenEdge session accesses the database through a server process. Multi-user client is the default connection mode on non-shared-memory systems and when you use the Host Name (-H) and Service Name (-S) parameters on shared-memory systems. All remote (over a network) connections access the database through a database server and are in multi-user client mode.

- **Multi-user direct access** — The current OpenEdge session accesses the database directly using shared memory. A process connected in multi-user direct-access mode is a self-service client. Direct access is available only on shared-memory systems and is the default connection mode on these systems. For information about shared-memory configurations, see *Configure OpenEdge*.

  You cannot connect to remote databases in multi-user direct-access mode.

When you connect to a database with an OpenEdge multi-user startup command, each database you specify on the command line is connected in the default multi-user mode. Each database you specify in a startup file or in a `CONNECT` statement in an application is also connected in the default multi-user mode for the current machine.

The first database specified after the OpenEdge single-user or OpenEdge single-user batch command is connected in single-user mode by default. Any additional databases specified with the Database Name (-db) connection parameter are connected in the default multi-user mode for the system. Use the Single-user Mode (-1) startup parameter to override the default multi-user connection mode for a database and connect to the database in single-user mode.

For more information about the Host Name (-H), Service Name (-S), Database Name (-db), and Single-user Mode (-1) startup parameters, see *Startup Command and Parameter Reference*.

Connection parameters

Connection parameters control how you connect to databases and are a subset of the OpenEdge startup parameters. All of the database connection techniques provide a way for you to specify connection parameters.

You can use a parameter file to specify connection parameters with any of the connection techniques. A *parameter file* is an operating system file that contains one or more startup parameters. A parameter file has a `.pf` extension. Use the Parameter File (`-pf`) startup parameter to invoke a parameter file.

Parameter files are an important factor in developing database connection strategies for multi-database applications. For example, you can use parameter files to maintain the startup parameters for a particular database, group of users, or system configuration.

For more information about startup parameters and parameter files, see *Startup Command and Parameter Reference*.

Connection techniques

The following are the five ways to connect a database:

- As an argument when starting OpenEdge
- With the `CONNECT` statement (in the Procedure Editor or in an ABL procedure)
- With the OpenEdge Data Dictionary
- With the OpenEdge Data Administration tool
- Using the Auto-connect feature

Note: Starting a server or broker for a database is distinct from connecting to a database. You must start a server or broker for a database before you can connect to the database in multi-user mode. For information about starting a server or broker, see *Configure OpenEdge*.

This section explains how to connect to databases using these various techniques. *OpenEdge External Programming Interface Reference* explains why you might choose one technique instead of another.

Connecting at startup

You can connect to databases at session startup with one of the OpenEdge startup commands or with an application startup file. For example, the following PRO command starts a single-user session and connects to three databases:

```
pro -db mydb1 -db mydb2 -db mydb3
```

Note: To retain compatibility with earlier versions, OpenEdge does not require you to specify the Database Name (-db) connection parameter for the first database on the command line.

If you specify more than one database when you start a session, it is important to specify the connection parameters for each database directly after the database name to which they apply, and before the next Database Name (-db) connection parameter. OpenEdge applies database connection parameters only to the previously specified database.

The following syntax illustrates the correct location for database connection parameters:

Syntax

```
mpro [ -db ] db1-name [db1-parameters]    [ -db db2-name [db2-parameters] ] . . .
```

You can specify all other parameters anywhere on the command line. If you specify the same parameter more than once, OpenEdge uses the value you specified for the last instance of the parameter.

You can also use a parameter file to specify database connection parameters. The following example shows a UNIX startup script for a client application where the last line invokes the _progres executable:

```
DLC=${DLC-/usr/dlc}; export DLC
PATH=:$PATH:$DLC; export PATH
PROPATH=:$DLC; export PROPATH
exec $DLC/_progres -pf parm1.pf -pf parm2.pf -p $APPL/start.p
```

The previous startup script sets up environment variables for an application and starts an ABL client application session using two parameter files called parm1.pf and parm2.pf. The parm1.pf parameter file contains the following entry:

```
-db appldb1 -1 bi-file1
```

The parm2.pf parameter file contains the following entry:

```
-db appldb2 -1 bi-file2
```

This example illustrates a common approach to connecting multiple databases when you start OpenEdge using a parameter file for each database. Each parameter file is specified on the command line with the Parameter File (-pf) startup parameter.

Connecting from a procedure with the CONNECT statement

The CONNECT statement allows you to connect to a database from an ABL procedure or from the Procedure Editor.

The following CONNECT statements connect the appldb1 database in single-user mode, the appldb2 database in multi-user mode, and the databases specified in the parameter file, parm3.pf (respectively):

```
CONNECT appldb1 -1.
CONNECT appldb2.
CONNECT -pf parm3.pf.
```

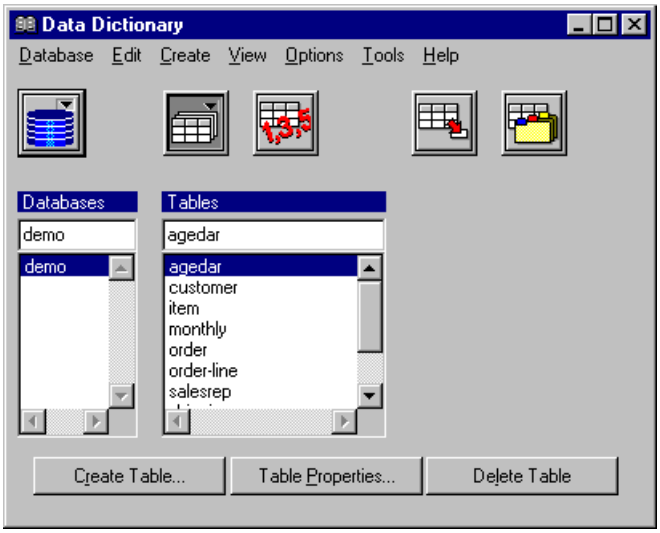
Connecting with the CONNECT statement is very similar to connecting at startup. Although it is possible to connect to several databases within one CONNECT statement, it is a good idea to connect only one database per CONNECT statement. A connection failure for one database causes a termination of the current CONNECT statement, leaving any subsequent database specified with the CONNECT statement unconnected. For detailed information about the CONNECT statement, see *OpenEdge External Programming Interface Reference*.

Connecting with the Data Dictionary

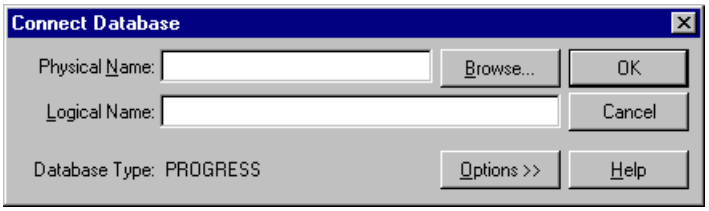
You can connect to a database during an OpenEdge session using the Data Dictionary.

To connect to a database with the Data Dictionary:

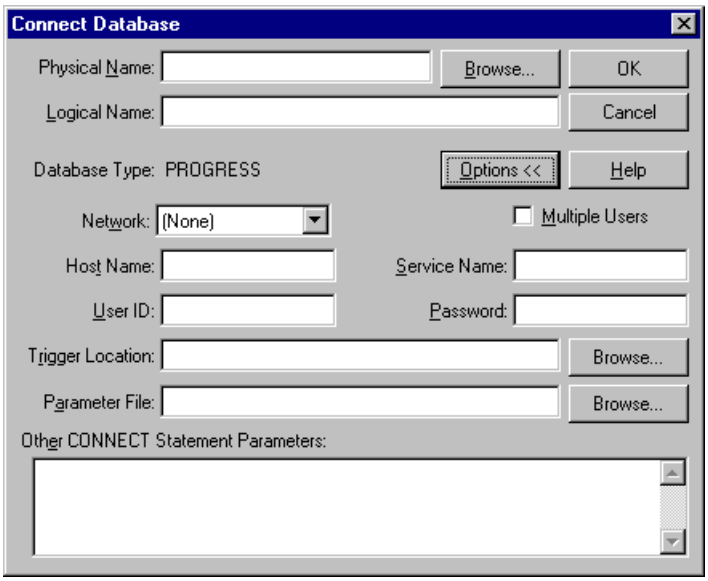
1. Open the **Data Dictionary** from the Windows **Start** menu or from the **Tools** menu of OpenEdge tools, such as AppBuilder and Progress Developer Studio for OpenEdge. If you already have a database connected, for example, the demo database, the following window appears:



2. Choose **Database** > **Connect**. The **Connect Database** dialog box appears:



3. Choose the **Options** button. The **Connect Database** dialog box expands to show optional connection controls:



4. Enter the following information:

   - **Physical Name** — Specifies the actual name of the database on a disk.
   - **Logical Name** — Specifies the database name that references a connected physical database.
   - **Database Type** — Specifies the database type. The only possible value is PROGRESS.
   - **Network** — Specifies the network type, if you are connecting to a network server. The only possible value is TCP.
   - **Multiple Users** — Specifies whether you want multiple users to be able to access this database simultaneously.
   - **Host Name** — Specifies the database server machine in a network environment.
   - **Service Name** — Specifies the broker or server service name in a network environment.
   - **User ID** — Identifies your user ID.
   - **Password** — Identifies your password.
   - **Trigger Location** — Identifies a directory or an ABL procedure library where trigger code is stored.
   - **Parameter File** — Specifies the parameter filename that contains the startup parameters for the database.
   - **Other CONNECT Statement Parameters** — Specifies any other startup parameters for the database that are not included in the parameter file.

5. Click **OK**.

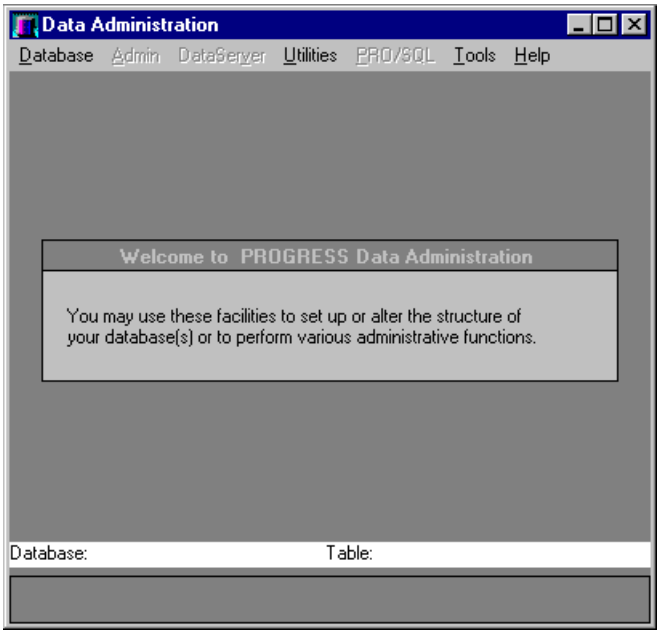
OpenEdge returns you to the Data Dictionary main window. The Data Dictionary constructs (and executes) a `CONNECT` statement using the information you supply. Therefore, any rules that apply to the `CONNECT` statement also apply to database connections using the Data Dictionary.

Connecting with the Data Administration tool

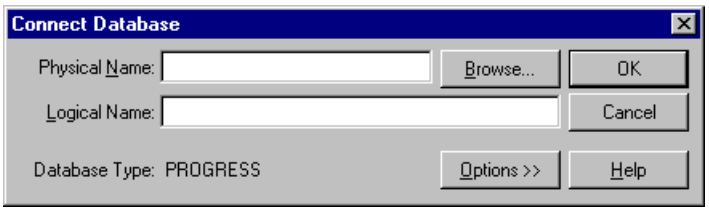
You can also connect to a database during an OpenEdge session using the Data Administration tool.

connect to a database with the Data Administration tool:

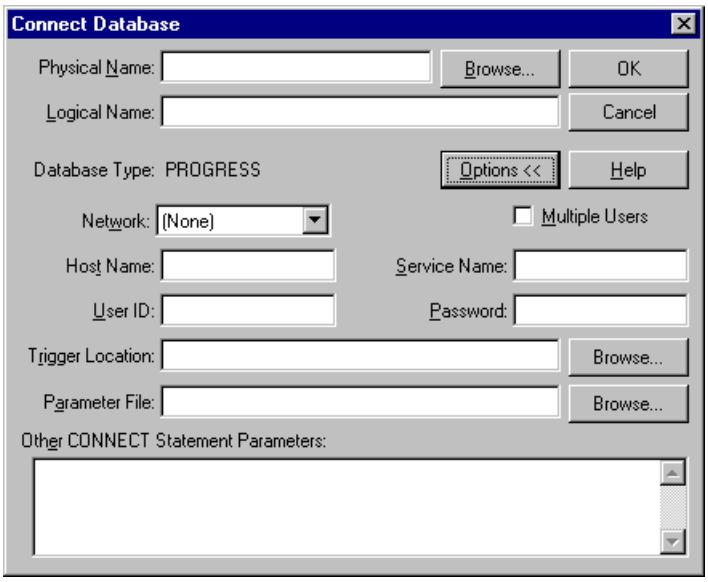
1. Open the **Data Administration** tool from the Windows **Start** menu or from the **Tools** menu of OpenEdge tools, such as AppBuilder and Progress Developer Studio for OpenEdge. The following window appears:



2. Choose **Database** > **Connect**. The **Connect Database** dialog box appears:



3. Choose the **Options** button. The **Connect Database** dialog box expands to show optional connection controls:



4. Enter the following information, then click **OK**:

   - **Physical Name** — Specifies the actual name of the database on a disk.
   - **Logical Name** — Specifies the database name that references a connected physical database.
   - **Database Type** — Specifies the database type. The only possible value is PROGRESS.
   - **Network** — Specifies the network type, if you are connecting to a network server. The only possible value is TCP.
   - **Multiple Users** — Specifies whether you want multiple users to be able to access this database simultaneously.
   - **Host Name** — Specifies the database server machine in a network environment.
   - **Service Name** — Specifies the broker or server service name in a network environment.
   - **User ID** — Identifies your user ID.
   - **Password** — Identifies your password.
   - **Trigger Location** — Identifies a directory or ABL procedure library where trigger code is stored.
   - **Parameter File** — Specifies the parameter filename that contains the startup parameters for the database.
   - **Other CONNECT Statement Parameters** — Specifies any other startup parameters for the database that are not included in the parameter file.

   OpenEdge returns you to the Data Administration main window. The Data Administration tool constructs (and executes) a `CONNECT` statement using the information you supply. Therefore, any rules that apply to the `CONNECT` statement also apply to database connections using the Data Administration tool.

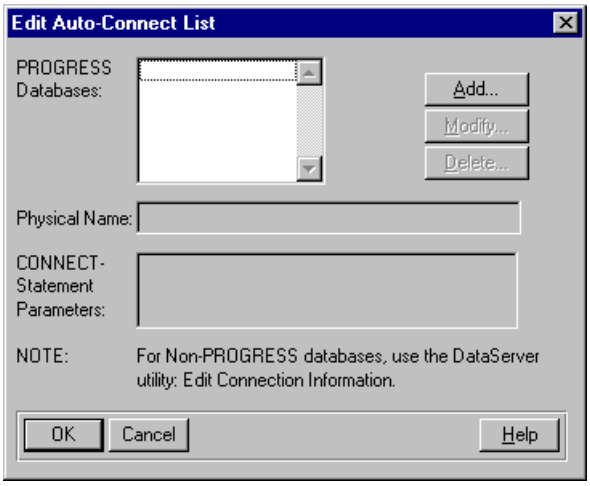
Connecting with auto-connect

The OpenEdge auto-connect feature allows you to connect to databases automatically as required during program execution. To perform an auto-connect operation, OpenEdge uses information stored in a primary database to connect to a secondary application database, before running compiled procedures that access the second database. The primary database (the one that contains the auto-connect information for one or more additional databases) must be connected before OpenEdge can perform an auto-connect.

Use the Data Administration tool to build an auto-connect list for a primary database.

To create or to edit an auto-connect list:

1. Access the **Data Administration** tool.
2. Choose **Edit Progress Auto-Connect ListUtilities**. **The Edit Auto-Connect List** dialog box appears:



3. Enter or edit connection information for a database.

If you connect to a database with the `CONNECT` statement, and that database also has an auto-connect entry in an already connected database, the connection information from both the `CONNECT` statement and the auto-connect list is merged. In this situation, the connection information in the `CONNECT` statement takes precedence. For more information about the `CONNECT` statement, see *ABL Reference*.

If the primary database is a schema holder for a DataServer schema, the DataServer schema is automatically included in the auto-connect list, but it does not appear on the auto-connect list as viewed from the Data Dictionary.

Connecting to a non-OpenEdge database

You can also use the OpenEdge tools to connect to a non-OpenEdge database.

To connect to a non-OpenEdge database:

1. Create an OpenEdge RDBMS. This database stores the target database's structure as an OpenEdge schema; it is a *schema holder*. A schema that corresponds to a non-OpenEdge database is a *DataServer schema*.
2. Connect the schema holder.
3. Use the appropriate Create Schema parameter from the Data Dictionary's database menu to create the DataServer schema in the schema holder. This parameter:

- Creates a _Db table record (auto-connect entry) for the non-OpenEdge database in the schema holder database
- Connects to the non-OpenEdge schema (through a DataServer)
- Displays the menu so that you can choose the tables that you want to include in the OpenEdge schema

4. Connect the DataServer schema.

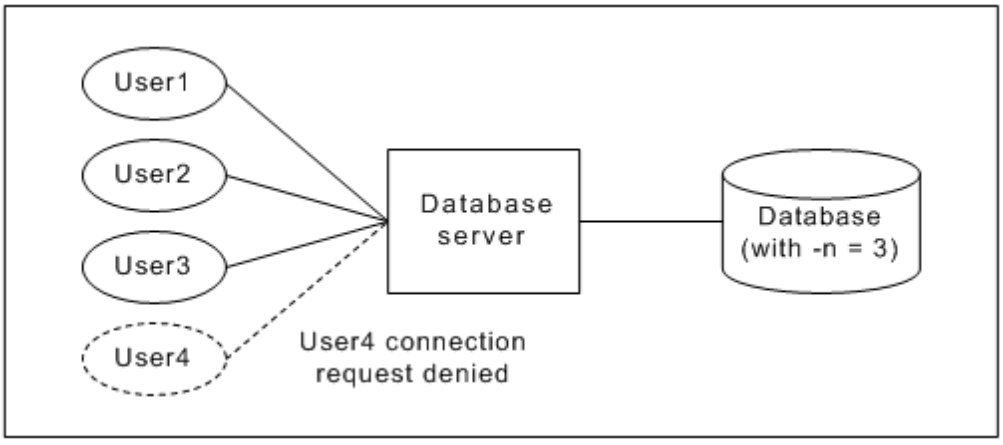
   For complete information about connecting and using non-OpenEdge databases, see the OpenEdge DataServer guides (*Use the Microsoft SQL Data Server* or *Use the Oracle Data Server*).

Connection denials when the user count is exceeded

When a user attempts to connect to a database using either the command line or a `CONNECT` statement and exceeds the maximum number of users allowed to connect to the database, as specified with the Number of Users (`-n`) startup parameter, OpenEdge denies the connection request and displays an error message. OpenEdge also records the connection denial in the database log file.

The following figure shows a connection denial when a connection request exceeds the maximum number of users. In this example, the Number of Users (`-n`) startup parameter allows a maximum of three user connections. When the fourth user attempts to connect, OpenEdge denies the connection.

Figure 1: Connection denial when the user count is exceeded



When OpenEdge denies a user connection because the maximum number of users is exceeded, you must terminate an existing connection before the user can establish a new connection.

You can use database usage reporting features to track user connections to databases. For information about database usage reporting features, see *Manage the OpenEdge Database* and *Configure OpenEdge*.

For more information about the Number of Users (-n) startup parameter, see *Startup Command and Parameter Reference*. For information about coding applications to handle connection denials, see *OpenEdge External Programming Interface Reference*.

Reducing connection times

To perform database activities, the ABL client keeps a copy of the database schema, called the *schema cache*, in memory. By default, OpenEdge creates the schema cache by reading the database schema stored in the database file. The client reads the schema in two parts:

- During connection, the client reads all the _Db and _File records.

- When the database is referenced, the client reads all the _Index, _Index-field, _File-trig, and _Field-trig records.

The time required to read the schema is usually minimal. However, the time required to read the schema might be unacceptable under the following conditions:

- If the client connects to the database over a wide area network.
- When a large number of clients connect to a database simultaneously. For example, after a database shutdown or crash.

To reduce the amount of time required to connect, OpenEdge lets you store the schema cache as a binary file, called a *schema cache file*, on a local disk. The client can then read the schema directly from the schema cache file.

To create the schema cache file, you build the desired schema cache and save it to a binary file using the ABL statement `SAVE CACHE`. The schema cache file is portable across systems, so you can create the file once and distribute it across a heterogeneous network of systems. For information about building and saving the schema cache file, see *OpenEdge External Programming Interface Reference*. For information about the `SAVE CACHE` statement, see *ABL Reference*.

To connect using the local schema cache file, specify the Schema Cache File (`-cache`) startup parameter when you connect to a database. If the schema cache file is valid, OpenEdge reads the schema from the local file instead of from the database. The schema cache is valid if the time stamp of the schema cache file matches the time stamp in the database master block. If the time stamps do not match, or for some reason OpenEdge cannot read the file, OpenEdge issues a warning message and reads the schema from the database. For information about the Schema Cache File (`-cache`) startup parameter, see *Startup Command and Parameter Reference*.

Starting clients without connecting to a database

To permit maximum flexibility for your application, you might want to start OpenEdge without connecting to a database. For example, you can start OpenEdge and name a startup procedure, such as a main menu procedure. This startup procedure (or a procedure that it calls) can connect to a database based on an end user's menu selection. To start OpenEdge without connecting to a database, execute the OpenEdge startup command without specifying a database name. On UNIX, use the `PRO` command; in Windows, use the `PROWIN32` command. For more information about startup commands, see *Startup Command and Parameter Reference*.

Selecting a working database for client use

A *working database* is the database that you are currently accessing through the Data Dictionary or the Database Administration tool. The working database is also known as the *active database*.

Selecting a database with the Data Dictionary

You can select a working database with the Data Dictionary when using either a graphical or character interface.

To select a working database with the Data Dictionary when using a graphical interface:

1. Access the **Data Dictionary**. The **Data Dictionary** displays the connected databases and their tables and fields.
2. Select the **database** you want for the working database.

Selecting a working database with the Data Dictionary when using a character interface

To select a working database with the Data Dictionary when using a character interface:

1. Access the **Data Dictionary**. The **Data Dictionary** appears.
2. Choose **Database** > **Select a Working Database**. OpenEdge lists the connected databases.
3. Select the new working database, then press **ENTER** to return to the **Data Dictionary** main window.

Selecting a database with the Data Administration tool

You can select a working database with the Data Administration tool only when using a graphical interface.

To select a working database with the Data Administration tool:

1. Access the Data Administration tool.
2. Choose **Database** > **Select a Working Database**. OpenEdge lists the connected databases.
3. Select the new working database, then choose **OK** to return to the **Data Administration** main window.

Disconnecting databases for clients

By default, OpenEdge disconnects all connected databases at the end of a session. To explicitly disconnect a database from an ABL client application, use the `DISCONNECT` statement. For more information about the `DISCONNECT` statement, see *ABL Reference* and *OpenEdge External Programming Interface Reference*.

Note: When you have more than one database connected and you disconnect the working database, OpenEdge automatically switches to the next connected database on the list.

Using the Data Dictionary to disconnect

You can disconnect a database with the Data Dictionary when using either a graphical or character interface.

To disconnect a database with the Data Dictionary when using a graphical interface:

1. Access the **Data Dictionary**. The Data Dictionary displays the connected databases and their tables and fields.
2. Select the database you want to disconnect from the database selection list.
3. Choose **Database** > **Disconnect**.
4. At the prompt, verify that you want to disconnect the database.

Disconnecting a database with the Data Dictionary when using a character interface

To disconnect a database with the Data Dictionary when using a character interface:

1. Access the **Data Dictionary**.
2. Choose **Database** > **Disconnect**. OpenEdge lists the connected databases.

3. Select the database you want to disconnect, then press **ENTER**.
4. At the prompt, verify that you want to disconnect the database.

Using the Data Administration tool to disconnect

You can disconnect a database with the Data Administration tool only when using a graphical interface.

To disconnect a database with the Data Administration tool:

1. Access the **Data Administration** tool.
2. Choose **Database** > **Disconnect**. OpenEdge lists the connected databases.
3. Select the database you want to disconnect, then choose **OK**.
4. At the prompt, verify that you want to disconnect the database.

3

Maintaining Application Security

OpenEdge® applications can use two basic types of user authorization: *application authorization*, which prevents unauthorized ABL application authenticated users from running application procedures and using other types of application resources, and *database authorization*, which prevents unauthorized ABL application authenticated users from modifying database tables and fields. (When a database is initially created, every user is a security administrator.) Application authorization always functions at run time using developer-defined privileges.

OpenEdge provides four types of database authorization: *compile-time authorization* ensures that only authorized users can compile procedures that access protected tables and fields at compile time; *run-time authorization* ensures that only OpenEdge authorized users can access database tables and fields when any ABL session (`.p` or `.r`) runs; *connection authorization* ensures that only OpenEdge authenticated users can connect to an OpenEdge database; and *security authorization* ensures that only authorized security administrators can manage table and field permissions or the records contained in certain security-related meta-schema tables.

OpenEdge also relies on security mechanisms at the operating system level to ensure that only authorized users access r-code, procedure libraries, and database files.

For information about establishing and maintaining connection security, schema security, and database file security, see *Manage the OpenEdge Database*. In addition to the security features described in this chapter, OpenEdge supports secure connections between ABL (Advanced Business Language) client and server components on the network using the Secure Sockets Layer (SSL). For more information, see *Introduction to Security and Auditing*.

This chapter provides details about user authorization, database authorization, and other security features in the following sections:

For details, see the following topics:

- Database table- and field-level security
- Compile-time security
- Run-time security

- Operating system security
- Designating a security administrator

Database table- and field-level security

OpenEdge allows you to set up table- and field-level security, which OpenEdge uses to authorize access to database tables and fields. This authorization can be specified to apply at:

- **Compile time** — Checks to verify that the user has authorization to compile procedures that access database tables and fields as coded. Database authorization is always applied at compile time. For more information, see Compile-time security on page 46.
- **Run time** — Checks an authenticated user's authorization to access protected databases tables and fields as defined by the database's security settings. You can choose to apply database authorization at run time, but it is not applied by default. For more information, see Specifying run-time permissions checking on page 41 and Run-time security on page 47.

Caution: If you plan to use OpenEdge SQL, as your **first task** be sure that you create an SQL DBA account as the **first** user account. Grant the account DBA rights by executing the GRANT statement. For more information, see *Develop SQL for OpenEdge*.

Note: Table- and field-level security for the `_Sequence` metaschema table applies at both compile time and run time. For more information on this table, see *OpenEdge External Programming Interface Reference*.

To set up database table- and field-level security, you and any other trusted users must be set up as security administrators. After you have designated yourself and others as security administrators, be sure that the user ID of each security administrator is included in all the permissions lists for the tables and fields for the database. For example, if you assign user ID sjones as a security administrator, but sjones is not included in the permissions for the customer table, sjones will not be able to modify permissions for the customer table. Any attempt to do so results in the error message, "Invalid access to change security for customer table." Thus, to modify table and field permissions, a user ID must be designated as a security administrator **and** be included in each of the CRUD (Create, Read, Update, and Delete) operations of the individual table and field permissions.

For more information about designating a security administrator, see Designating a security administrator on page 48.

The user ID format you can specify for table- and field-level security differs depending on the tenancy configured for your database. So, if your database is:

- **Non-multi-tenant (the default)** — The user IDs that you specify must be non-qualified, and cannot contain the domain-name delimiter character (`@`). OpenEdge interprets the user ID as a user name only.
- **Multi-tenant (license required)** — The user IDs that you specify can be fully qualified, containing both a user and domain name separated by the domain-name delimiter character (`@`). If you specify a non-qualified user ID, OpenEdge interprets it as the specified user name in the blank (`""`) domain.

Note that to authenticate their identity for any database connection (non-multi-tenant or multi-tenant), users can authenticate to OpenEdge using a fully qualified user ID, which tells OpenEdge the domain in which to authenticate their identity. However, for non-multi-tenant databases, you can specify only a user's non-qualified user ID (user name) in table- and field-level security permissions lists.

For more information about user authentication, domains, and database tenancy, see *OpenEdge Getting Started: Identity Management*.

Specifying run-time permissions checking

By default, database authorization is always applied at compile time, which does not protect access to the database against ABL `.p` or `.r` code compiled outside of your application. Optionally, you can specify that the table and field security permissions are checked and enforced by the OpenEdge run time for all ABL sessions, regardless of compile time authorization.

Note: Table and field security permissions for the `_Sequence` metaschema table are checked and enforced at both compile time and run time by default.

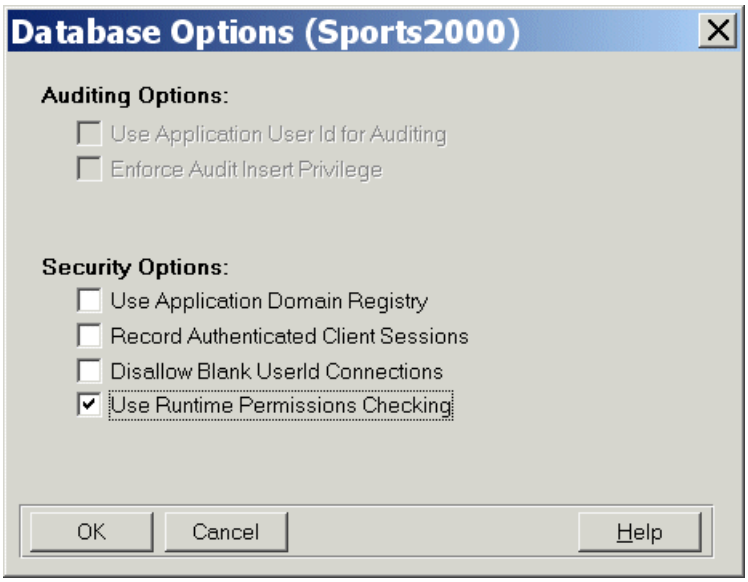
Database permissions are applied to an authenticated user whose identity is set for a database connection through the User ID (`-U`) and Password (`-P`) startup parameters and the following ABL functions and method:

- `SECURITY-POLICY:SET-CLIENT()` method
- `SET-DB-CLIENT()` ABL function
- `SETUSERID()` ABL function

For more information on these security startup parameters, functions, and methods, see *OpenEdge Getting Started: Identity Management* and *OpenEdge External Programming Interface Reference*.

To include run-time permissions checking:

1. Connect to the database.
2. Access the **Data Administration** tool if you are using a graphical interface or the Data Dictionary if you are using a character interface.
3. Choose **Database Options** > **Admin**. The **Database Options** dialog box appears.
4. Check the **Use Runtime Permissions Checking** option, as shown:



5. Click **OK**.

   Now when you set database table and field permissions, they will apply at run time in addition to compile time.

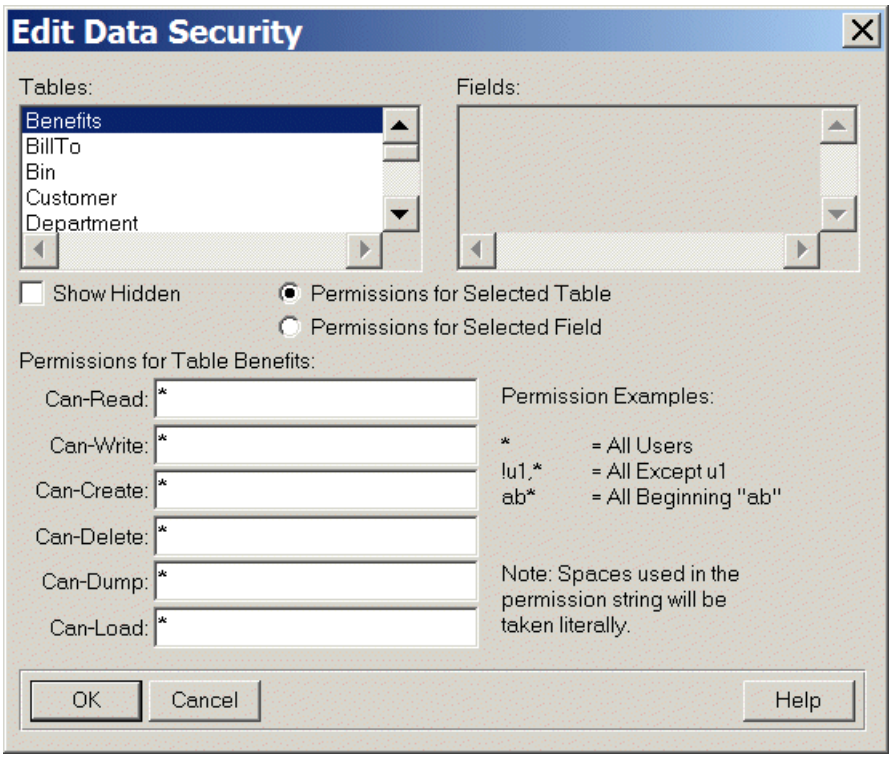
Setting table and field permissions

You can set table and field permissions by using the Change/Display Data Security utility in the Data Dictionary or the Edit Data Security menu option in the Data Administration tool. Note that once security administrators are designated, only they can access this utility. All other users are denied access by the tools with the message, "You must be a Security Administrator to execute this function." For information on designating security administrators, see Designating a security administrator on page 48.

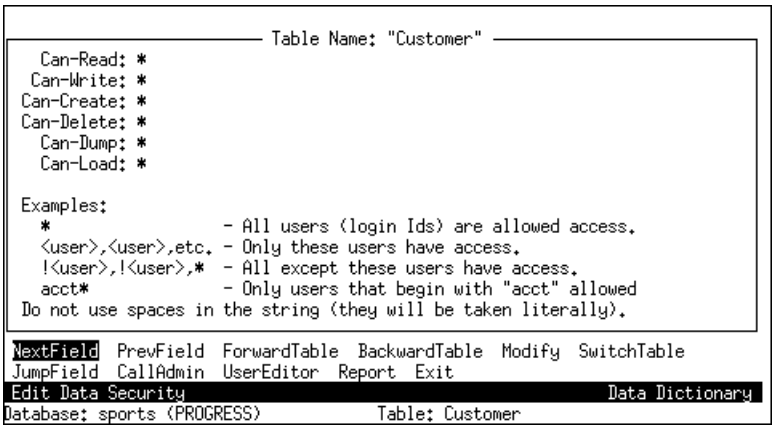
To set table or field permissions:

1. Access the **Data Administration** tool if you are using a graphical interface or the Data Dictionary if you are using a character interface.
2. Choose **Admin** > **Security** > **Edit Data Security**.

   If you are using a graphical interface, the **Edit Data Security** dialog box appears:



   If you are using a character interface, OpenEdge alphabetically lists the tables defined for the working database. Choose the table for which you want to specify permissions. The **Edit Data Security** dialog box appears:



3. Choose the table or field for which you want to specify permissions.

   If you are using a graphical interface, use the **Permissions for Selected Table** and **Permissions for Selected Field** radio buttons, then choose the table or field from the selection lists.

   If you are using a character interface, use the options at the bottom of the dialog box to choose the table or field.

4. Specify the security permissions for the table or field. The following tabel lists the security permissions you can specify for a table or field.

   Table 2: Table and field security permissions

Permission	Description
Can-Read	Specifies the users who have permission to read a table or field.
Can-Write[1]	Specifies the users who can write to a table or field or update records.
Can-Create	Specifies the users who can create new records. The user with Can-Create privileges automatically has Can-Write privileges.
Can-Delete	Specifies the users who can delete records from a table.
Can-Dump	Specifies the users who can bypass dump trigger security by using DISABLE TRIGGER.
Can-Load	Specifies the users who can bypass load trigger security by using DISABLE TRIGGER.

   Security permissions consist of a list of one or more comma-separated user ID patterns.

   The following table lists basic types of expressions you can use to define the permissions for a table or field. This table shows permissions using non-qualified user IDs, as you specify for a non-multi-tenant database.

Note: An empty permission string denies the table and field permissions to **every** user and is not recoverable.

---

[1] Setting this permission requires that the Can-Read permission also be set.

Table 3: Basic expressions for table and field permissions

Expression	Meaning
`[,]<blank>,...`	The blank user ID has access.
`*`	All user IDs have access.
`mark`	The user ID `"mark"` has access.
!mark	The user ID `"mark"` does not have access. **Note:** The `!` permission negation is useful only when the permission list ends with `",*"`
,!mark,*	The blank user ID and the user ID `"mark"` do not have access.
`mark*`	All user IDs that begin with `"mark"` have access.
`*jones`	All user IDs that end with `"jones"` have access.
`db*user`	All user IDs that begin with `"db"` and end with `"user"` have access.

The following table shows permissions using fully qualified user IDs, as you specify for a multi-tenant database.

Table 4: Table and field permissions using fully qualified user IDs

This expression . . .	Permits access to tables and fields from . . .
	The blank user name in the blank domain
`*`	All user ID's
`mark@acme.com`	The `"mark"` user name in the `"acme.com"` domain
`mark`	The `"mark"` user name in the blank domain
`mark*`	All user names that begin with `"mark"` in the blank domain
`*jones`	All user names that end with `"jones"` in the blank domain
`db*user`	All user names that begin with `"db"` and end with `"user"` in the blank domain
`@acme`	The blank user name in the `"acme"` domain
`*@acme`	All user names in the `"acme"` domain
`@`	The blank user name in the blank domain
`mark@*`	The `"mark"` user name in any domain

This expression . . .	Permits access to tables and fields from . . .
@	Any user name in any domain
mark*@acme.*	Any user name that begins with "mark" in any domain that begins with "acme."
@.admins	Any user name in any domain that ends with ".admins"

For a more detailed description of the options available for specifying these permissions, see the sections on configuring and implementing authorization in *OpenEdge Getting Started: Identity Management*.

Any changes you make to the table permissions do not affect your current session or any other current sessions. This means that if other users are working while you change table permissions, they are not affected. To use the new permissions, you must exit and restart the OpenEdge ABL session.

Note: Do not try to bypass the Data Dictionary or Data Administration tools to modify the permissions for the tables and fields in the database. You might lock yourself out of the database.

Example of personnel security

This example shows how to define permissions for a sample multi-tenant database. Suppose the user IDs are joesmith@acme, jenjones@acme, joesmith@gentac, and tracyblack@gentac.

Suppose you want all users to have Can-Read permission for a table, but only certain users to have Can-Write, Can-Create, and Can-Delete privileges. The following example shows how to specify this information, in which all users have Can-Read permission; Joe Smith at Acme and Tracy Black at Gentac have Can-Write permission; all users at Acme and at Gentac have Can-Create permission; and both Joe Smiths, one at Acme and the other at Gentac, have Can-Delete permission:

```
Can-Read:    *
Can-Write:   joesmith@acme,tracyblack@gentac
Can-Create:  *@acme,*@gentac
Can-Delete:  joesmith@*
```

In the next example, the blank user ID does not have Can-Read permission, whereas all other users have the permission. All users, except Joe Smith at Acme and Joe Smith at Gentac, have Can-Write permission:

```
Can-Read:   !,*
Can-Write:  !joesmith@acme,!joesmith@gentac,*
```

Determining the privileges of the blank user ID

By not specifying the User ID (-U) and Password (-P) connection parameters when connecting to a database (at startup or in the CONNECT statement), the user can access the database with a default user identity that has the blank user ID or the OS process user ID.

You can also authenticate a blank ("") user ID for a database connection by connecting to a database (at startup or in the CONNECT statement) with the User ID (-U) and Password (-P) connection parameters with no value specified for -U, or by using an ABL function or method like SET-DB-CLIENT () to explicitly set a blank user identity for an existing database connection.

For more information on these connection parameters and ABL functions and methods for setting database connection identity, see *OpenEdge Getting Started: Identity Management* and *OpenEdge External Programming Interface Reference*.

Any user can use OpenEdge or an OpenEdge application with a blank user ID and access tables and fields in the database as long as the following conditions are all met:

- The database is configured to allow blank user ID connections (the default)
- The table- and field-level permissions permit it (blank user ID) (the default)
- The procedures being run are precompiled with table/field access authorized

As the security administrator, you can deny privileges to the blank user ID to ensure that users with blank user ID do not gain access to the data by following these steps.

Note: These steps must be performed **every time** a new table is added to the database.

To restrict blank user ID access privileges:

1. Access the **Data Administration** tool if you are using a graphical interface or the **Data Dictionary** if you are using a character interface.
2. Choose **AdminSecurityDisallow Blank Userid Access**. OpenEdge prompts you to verify that you want to prevent users with blank user IDs from accessing the working database.
3. Choose **Yes**. OpenEdge denies the user all security permissions and inserts an exclamation point (`!`) at the beginning of all the table and field permissions for the database. You can restore a blank user ID's access to selected tables and fields by modifying the permissions.

   Note: You can also restrict a blank user ID from connecting to a database by setting the **Disallow Blank UserId Connections** option in the **Database Options** dialog box. For more information, see the online help for the Data Administration tool and *Database Tools*.

Compile-time security

Compile-time security checking is built into OpenEdge. You define compile-time security for an application database at the table and field levels to prevent the user from writing their own procedures to access data in the database.

OpenEdge lets you define the type of access rights or permissions different users can have to the tables and fields in your database applications. OpenEdge checks these permissions when the user runs and compiles a procedure for the first time during an OpenEdge session. The default compile-time checking is useful for applications that are compiled each time a different user runs the application.

However, the default compile-time permissions apply to application code that is compiled each time a different user executes it and are not sufficient for precompiled applications. If you use CRC-based r-code (the default), the user can compile a procedure against a database that has the same schema as the database (a counterfeit database) and then run the procedure against the database. Since the default for OpenEdge is to do compile-time permission checking only, your database is unprotected at run time unless you do one of the following:

- Turn on run-time permission checking as described in Specifying run-time permissions checking on page 41.
- Use the PROUTIL utility's `DBAUTHKEY` qualifier to set an authorization key for the database. The authorization key prevents unwanted r-code that has been compile-time checked only from running against the database.

For more information about CRC-based object code, see R-code Features and Functions on page 189 For more information about the PROUTIL utility, see *Manage the OpenEdge Database*.

Run-time security

Run-time security can be supported both in an ABL application and in OpenEdge database table and field permissions settings.

Application support

A developer can write custom run-time authorization to prevent unauthorized users or precompiled procedures from accessing restricted database tables and fields. To establish run-time security for precompiled procedures, you must set up a permissions table within the database and ensure that only your application's precompiled procedures are allowed to run against the database.

The permissions table contains records that specify users who are authorized to run specific procedures or classes. Each record in the permissions table must contain at least two fields: an Activity field and a Can-Run field. The Activity field contains the name of the procedure and the Can-Run field contains the user IDs of those who have permission to run the procedure. Within the application, you can use the ABL `CAN-DO` function to test whether the current application user can run a specific procedure.

As security administrator, you must maintain the permissions table. It is also your responsibility as the developer to provide the tools to maintain this table. For more information about setting up run-time security with a permissions table, see the information on run-time security in *OpenEdge External Programming Interface Reference*.

OpenEdge support

To establish run-time security for tables and fields, you must set the table and field permissions using the Data Administration tool or character Data Dictionary.

As security administrator, you can also specify the run-time security permissions for database tables and fields. For more information, see Setting table and field permissions on page 42.

Operating system security

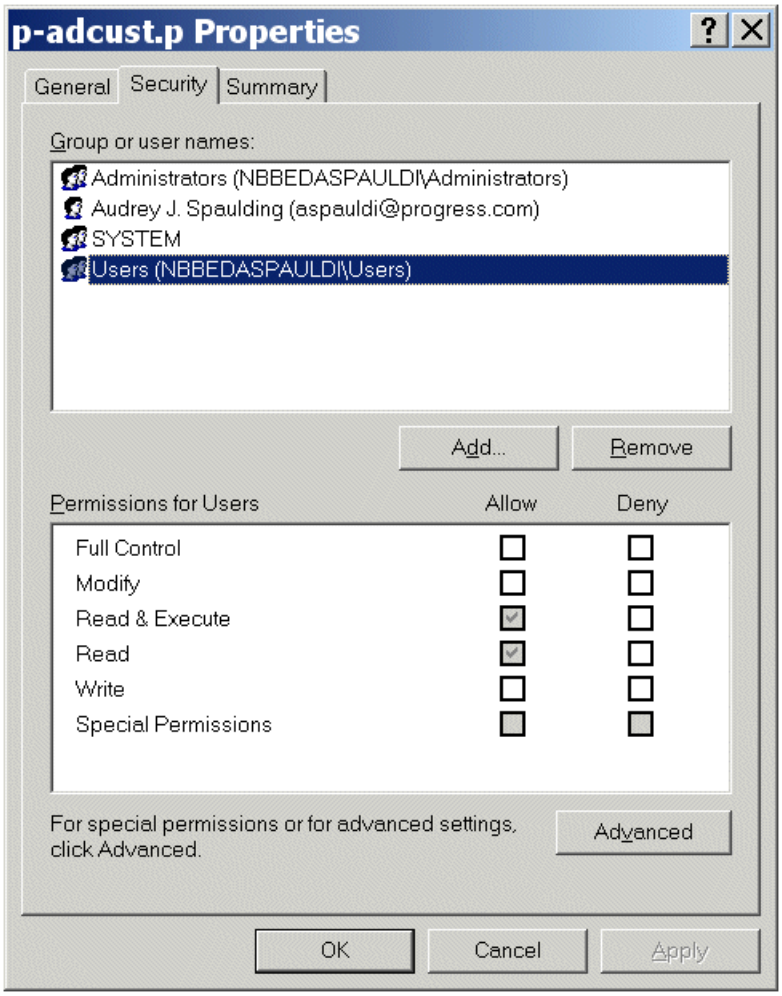
Each operating system provides security measures that you can use to protect r-code, procedure libraries, and database files for access by other OS user accounts and OS administrators.

On UNIX, you can use operating system file security to protect your procedure files in addition to, or instead of, including security checking in the application procedures. If you, as the application administrator, want to be the only user allowed Write access to `p-adcust.p` when everyone else has only Read access, you should create the source and r-code files with your user ID, then change the permissions for the files accordingly. To change the permissions for `p-adcust.p`, enter one of the following UNIX commands:

```
chmod 640 p-adcust.p
chmod 640 p-adcust.r
```

These permissions indicate that you are the only user allowed to read from (R) and write to (W) `p-adcust.p`. No other users are allowed any kind of access to that procedure.

In Windows, you can set authorization on a specific file or directory through the file or directory's properties. For example, if you want to set permissions for `p-adcust.p`, in Windows Explorer right-click on the file and select **Properties** from the pop-up menu. The **Properties** dialog box appears:



After you click the **Security** tab, you can specify various levels of authorization to the file. You can also click the **Advanced** button to set additional security options.

For more information about using the operating system to protect database files, see OpenEdge Data Management: Database Administration. For more information about setting operating system permissions, see your operating system documentation.

Designating a security administrator

Each level of security uses an authenticated user ID to determine the user's authority to access data and files. Associating a user ID with a password provides even more protection against unauthorized access.

The implementation of security for the main features of OpenEdge is the responsibility of administrators in the following areas:

- ABL security administration
- Auditing administration

- Transparent Data Encryption administration

ABL security administration

The ABL security administrator is responsible for maintaining the ABL security information in the database. When a database is first created, everyone is a default security administrator. However, you should assign user IDs as the security administrator. If you plan to define user IDs in the database `_User` table accounts, you must also designate at least one user from these accounts as a security administrator. Note that once you add at least one individual user ID as a security administrator, no other user ID can function as a security administrator for a future connection to this database.

Caution: Make sure you can log into the database with a user ID **before** you assign it as a security administrator.

Typically, the database administrator also serves as security administrator. The security administrator uses:

- The Data Dictionary, the Data Administration tool, or the Database Administration Console of OpenEdge Management or OpenEdge Explorer to establish connection, schema, and database access security at compile time or run time
- An activity table provided by you (the developer) to establish run-time authorization for accessing specified precompiled procedures or classes

---

Note: SQL has its own administrator (DBA) to grant access privileges. The SQL DBA cannot set ABL permissions, just as an ABL security administrator cannot grant SQL privileges. For more information, see OpenEdge Data Management: SQL Development.

For more information about designating a security administrator or establishing and maintaining user IDs, passwords, and data access privileges, see *Manage the OpenEdge Database* and *Introduction to Identity Management*.

Auditing administration

By default, OpenEdge applies a grant authorization model to all audit-related database tables. This means that in order for an individual to be able to create audit policies and manage audit data, the individual must be granted the appropriate privileges to do so.

Because you might not want only one individual to have responsibility for all audit-related activities, you can assign to certain users one or more auditing privileges. When you assign privileges to a user, you also decide whether that user can then grant the same privileges to other users. Only users who have been granted the appropriate privileges can perform the corresponding auditing functions.

There are four audit security privileges, one of which is known as the audit administrator. The audit administrator is an authenticated user who has been granted privileges to create, update, and delete audit policies. The audit administrator also has unrestricted read access to all the audit tables and is the only user authorized to configure audit policy.

For more information about audit security privileges, see *OpenEdge Getting Started: Core Business Services - Security and Auditing*.

Transparent data encryption administration

You must be connected to the database locally and have ABL security administrator and key store administrator privileges to create encryption policies with the Data Administration tool. For more information, see *Introduction to Security and Auditing*.

4

Maintaining User Environments

This chapter explains how to maintain user interface environments in Windows and on UNIX, as described in the following sections:

For details, see the following topics:

- Maintaining the Windows user environment
- Maintaining the UNIX user environment

Maintaining the Windows user environment

In the Windows software environment, OpenEdge uses the `progress.ini` file as well as the Registry to maintain environment variables.

When you install OpenEdge, the installation program automatically updates the Registry with the information in the `progress.ini` file that is shipped to you. Progress Software Corporation recommends that you maintain environment variables in both the `progress.ini` file and the Registry. If you modify environment variables in the `progress.ini` file, you must update the information in the Registry.

Although there are several methods for adding and deleting information from the Registry directly, Progress Software Corporation recommends that you do not use these direct update methods. Instead, maintain the Registry information by editing the `progress.ini` file and using the INI2REG utility to update the Registry. This approach synchronizes the information in the `progress.ini` file and the Registry without causing unintended edits to the Registry.

For more information about modifying the `progress.ini` file, see the Maintaining the progress.ini file on page 56.

In addition to the `progress.ini` file, the OpenEdge installation includes a Windows manifest file which provides your ABL (Advanced Business Language) application with the Windows XP look. For more information on this file, see Windows manifest file on page 80.

The OpenEdge GUI for .NET enables you to expose your ABL applications through a .NET interface. GUI for .NET applications have extra requirements of which you need to aware. For more information on maintaining an environment for GUI for .NET applications, see Environments with OpenEdge GUI for .NET applications on page 81.

Using the INI2REG utility

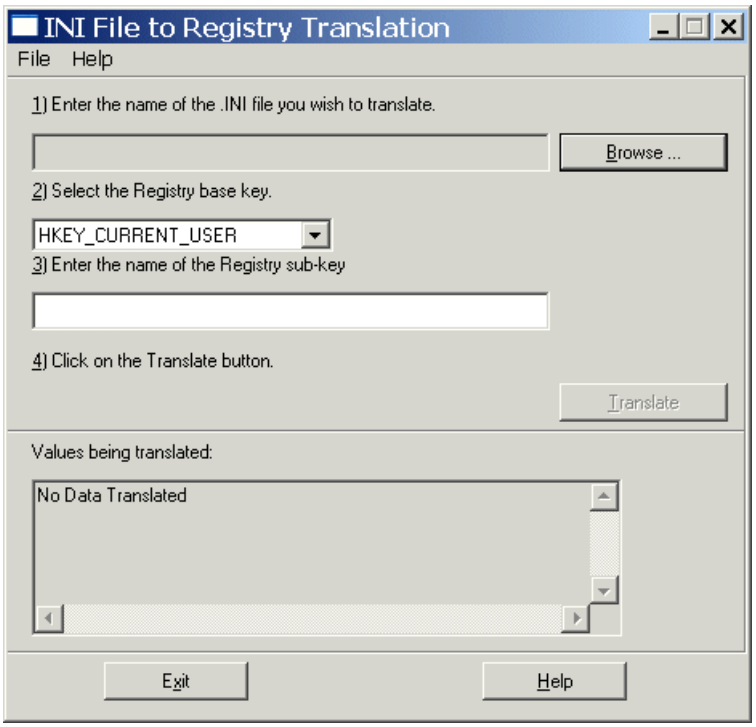
The INI2REG utility lets you translate the contents of an initialization (`.ini`) file into comparable Registry key entries. You can access the INI2REG utility through a graphical user interface or from the command line.

To access the INI2REG utility and translate an initialization file:

1. From the Windows **Start** menu, choose **OpenEdge** > **Proenv**.
2. Enter the INI2REG command as follows:

```
ini2reg
```

The **INI File to Registry Translation** dialog box appears:



3. From the **INI File to Registry Translation** dialog box, choose the **Browse** button or **File** > **Open**. The **Open** dialog box appears.
4. Type in or browse for the initialization file you want to translate, and choose **OK**. The name of the selected file appears in the `.INI` file to translate field.
5. From the **Registry base key** drop-down list, select the base key under which you want to create the subkey entries. The default subkey that corresponds to the selected initialization file and base key appears in the **Registry subkey** field.

If you want to use a subkey other than the default, type the subkey name in the **Registry subkey** field.

6. To begin the translation, choose the **Translate** button or **FileTranslate**. For each entry in the initialization file, INI2REG creates a Registry key entry with the same value. As each entry is translated, it appears in the **Values** being translated display area. If a Registry key entry already exists, INI2REG prompts you to overwrite it.
7. To exit, choose the **Exit** button or **FileExit**.

Using the command line interface

To access the INI2REG utility and translate an initialization file from the command line, specify the INI2REG command using the following syntax:

Syntax

```
ini2reg { -a | -ao } [ -i ini-file-name] [ -b base-key] [ -s subkey] [ -d
default-subkey]
```

`-a`

Automates the translation. That is, it processes the translation automatically without presenting the graphical user interface.

`-ao`

Automates the translation and automatically overwrites existing Registry entries.

`-i` *`ini-file-name`*

The pathname of the initialization file to translate.

`-b` *`base-key`*

The base key under which the subkey is created. The value of `base-key` must be one of following:

- `HKEY_CURRENT_USER` (default).
- `HKEY_USERS`
- `HKEY_LOCAL_MACHINE`
- `HKEY_CLASSES_ROOT`
- `HKEY_CURRENT_CONFIG`
- `HKEY_DYN_DATA`

`-s` *`subkey`*

The subkey under which entries are created. The subkey is created under the base key specified in the -b parameter.

Use -d to specify a default subkey for the translation destination.

`-d default-subkey`

The default subkey under which entries are created. The default subkey is created under the base key specified in the -b parameter.

If the base key is `HKEY_LOCAL_MACHINE` or `HKEY_CURRENT_USER`, INI2REG prepends `SOFTWARE` to the specified default subkey. If the initialization file name is not `progress.ini`, INI2REG appends the root name of the initialization file to the specified default subkey. If this parameter is not specified, INI2REG uses a default subkey of `PSC\PROGRESS\`*`version`*.

You can generalize this as follows:

```
base-key\[SOFTWARE]\default-subkey\[rootname]
```

Use -s to specify a complete subkey for the translation destination.

By default, the INI2REG utility creates Registry key entries for 32-bit OpenEdge at the following location:

```
HKEY_CURRENT_USER\SOFTWARE\PSC\PROGRESS\version
```

By default, the INI2REG utility creates Registry key entries for 64-bit OpenEdge at the following location:

```
HKEY_CURRENT_USER\SOFTWARE\PSC\PROGRESS\x64\version
```

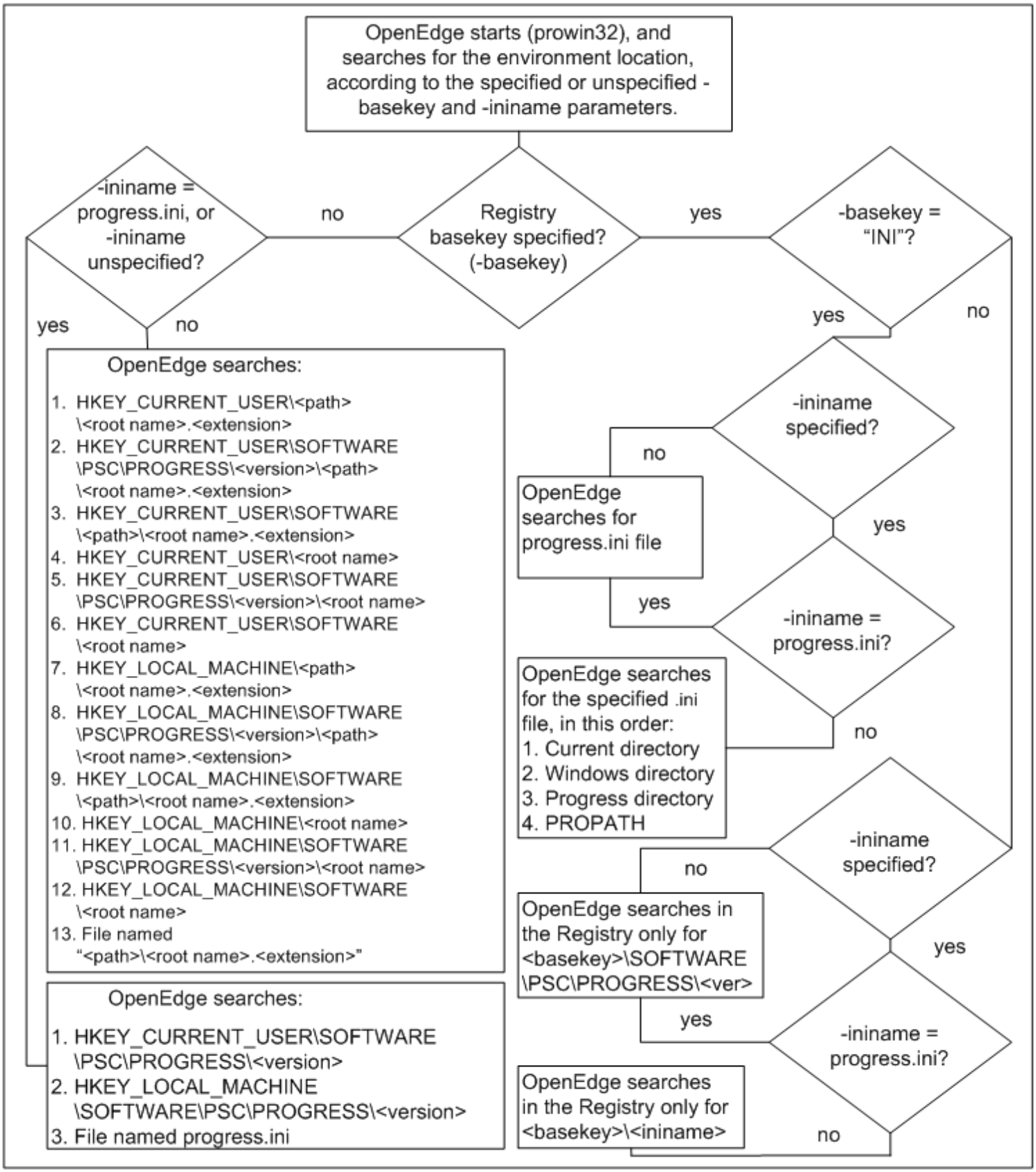
Searching progress.ini and the Registry at startup

At startup, OpenEdge searches for a Registry location, a `progress.ini` file, or both. The particular search path depends upon the parameters you specify with the `prowin32` command, particularly the Registry Base Key (`-basekey`) and Initialization File (`-ininame`) parameters. You can use these parameters to create a more efficient search at startup.

Note: The `-basekey` and `-ininame` parameters can only be used on the command line. You cannot use them within a parameter file.

The following illustrates the path that OpenEdge follows when searching for environment information in Windows.

Figure 2: Windows environment information search path



Note: OpenEdge searches for environment information in the HKEY_CURRENT_USER\SOFTWARE\PSC\PROGRESS\x64\<version> location if you have installed 64-bit OpenEdge.

When OpenEdge does not find environment information in the Registry, or when it was started with the -basekey INI startup parameter, it searches through directories for a progress.ini file in the following order:

1. The current working directory (as defined in the Working Directory field in the Program Item properties sheet for Windows)
2. The WINDOWS directory
3. The directory that contains the OpenEdge executable
4. The user's PROPATH

   This search order fosters deployment flexibility because you can:

   - Create a customized progress.ini file
   - Place the file in a consistent place on every machine
   - Allow all users to share the same file on a network

   After OpenEdge locates the environment information, OpenEdge reads the values for DLC, PROMSGS, and other environment variables. For each variable, OpenEdge first searches the location where it found the environment information. If OpenEdge does not find the variable, it searches for an operating system environment variable.

OpenEdge expects to find all environment settings in the same location it finds the environment variables. There is one exception: OpenEdge searches HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE first when it starts without the -basekey parameter and finds the environment information in the Registry. If OpenEdge writes a value to the Registry, however, it will always write it to HKEY_CURRENT_USER. This allows multiple users of a single machine to customize settings individually. The environment setting location could be either a Registry key or an initialization file, but not both.

The ABL GET-KEY-VALUE statement also searches for user environment information. The GET-KEY-VALUE statement searches either the Registry or a progress.ini file, but not both. When GET-KEY-VALUE searches the Registry, it searches both of the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE keys if the following conditions are true:

- The OpenEdge startup command (prowin32) used for the current session did not include the -basekey parameter.
- OpenEdge found the environment information in the Registry at startup or at the last LOAD statement.

The ABL PUT-KEY-VALUE statement writes to the location where the environment information was found at startup. If that location is the Registry, and if the prowin32 command did not include the -basekey parameter at startup, the PUT-KEY-VALUE statement writes to the HKEY_CURRENT_USER key and not to the HKEY_LOCAL_MACHINE key.

Maintaining the progress.ini file

The progress.ini file specifies environment variable settings for the Windows environment. It contains sections and settings for the following types of options:

- Options required by OpenEdge
- Options required by OpenEdge tools (such as the Procedure Editor and the Report Builder)
- User-defined options

- Environment variables

The following shows excerpts from a sample `progress.ini` file.

Sample OpenEdge initialization file in Windows

```
[Startup]
V6Display=no
;ImmediateDisplay=yes
;MultitaskingInterval=100
DefaultFont=MS Sans Serif, size=8
DefaultFixedFont=Courier New, size=8
DLC=C:\Progress\OpenEdge
Use-3D-Size=Yes
PROPATH=.,C:\Progress\OpenEdge\C:\Progress\OpenEdge\bin
UseSourceEditor=yes
EditorConfigPath=C:\OpenEdge\WRK\proedit
...[WinChar Startup]
DLC=C:\Progress\OpenEdge
PROCFG=C:\DLC\PROGRESS.CFG
PROMSGS=C:\DLC\PROLANG\PROMSGS.GER
PROPATH=.,C:\DLC
V6Keys=No
V6FKeys=No
...
```

When installed, the `progress.ini` file contains the following sections:

- Startup section on page 58
- Colors section on page 67
- Default Window section on page 68
- Fonts section on page 69
- WinChar Startup section on page 70
- WinChar Colors section on page 72
- WinChar Default Window section on page 73
- WinChar Keys section on page 73

The `progress.ini` file can also contain the following optional sections:

- Debug-Init section on page 74
- Debug-Macros section on page 75
- Debug-Buttons section on page 75
- Debug-Buttons section on page 75
- Keys section on page 75
- Keymap section on page 76

You can edit the `progress.ini` file to customize environments at different sites. The `progress.ini` file is located in the *`install-dir`*`\bin` directory, by default.

The following sections describe each of these `progress.ini` file sections.

Startup section

The Startup section of the `progress.ini` file contains options that specify the following environment characteristics for graphical user interface clients:

- Display attributes and multi-tasking capabilities
- Default fonts used for displaying and printing alphanumeric data

The Startup section also contains OpenEdge environment variables that you can set for use with graphical user interface clients. See Specifying environment variables on page 78 for a list of environment variables you can set in the Startup section.

The Startup section contains the following options:

`AlignFrameTitles`

```
AlignFrameTitles=n
```

Lets you specify the alignment of frame titles in a graphical user interface. Valid settings include the following: 0 (left justified), 1 (centered), and 2 (right justified). The default is centered.

`ButtonImageBorderMode`

```
ButtonImageBorderMode=n
```

Use this setting if you want to change the default behavior of how OpenEdge draws borders on buttons with images. The border drawing behavior for image buttons depends on two factors:

- The type of image button—single-image or dual-image. *Dual-image buttons* use two images (an "up" image and a "down" image). *Single-image buttons* use one image (an "up" image which is shifted slightly to mimic a "down" appearance for the button).
- The presence of a manifest file. For more details on manifest files, see Windows manifest file on page 80.

The default behavior is:

- For single-image buttons, OpenEdge creates a Windows XP-style border (rounded corners, no 3D shadow) when using the manifest file. It draws a Windows classic-style border (angular corners with 3D shadow) when not using the manifest file.
- For dual-image buttons, OpenEdge does **not** draw a border for the button, regardless of whether there is a manifest file. (Note that this is the default behavior. You can change the value of `ButtonImageBorderMode` to **2** if you want OpenEdge to draw a border for dual-image buttons.)

The following table describes OpenEdge's border drawing behavior, based on the value of the `ButtonImageBorderMode` setting. If you want to keep the default behavior, you do not need to add this setting to your `progress.ini` file.

Table 5: Image button border drawing behavior

Type of image button	ButtonImageBorderMode value of *n*		
	1 (default)	2	3
Single-image button with manifest file	Adds an XP-style border to the button	Adds an XP-style border to the button	Adds a classic-style border to the button
Single-image button without manifest file	Adds a classic-style border to the button	Adds a classic-style border to the button	Adds a classic-style border to the button
Dual-image button with manifest file	No border added	Adds an XP-style border to the button	No border added
Dual-image button without manifest file	No border added	Adds a classic-style border to the button	No border added

Note that `ButtonImageBorderMode` affects clients running on XP with a manifest file and clients running on non-XP systems (or XP without a manifest file). The same setting is required for all clients.

`CarefulPaint`

```
CarefulPaint={ YES | NO }
```

Lets you set the accuracy with which OpenEdge repaints overlapping 3D widgets and frames. Set to `YES` for OpenEdge to repaint overlapping 3D widgets and frames slowly and carefully. Set to `NO` for OpenEdge to repaint overlapping 3D widgets and frames quickly (which might reduce the quality of the graphic). The default value is `YES`.

This option applies only when `UseNative3D=NO`.

`ClientTimeOut`

```
ClientTimeOut=n
```

Lets you specify the number of minutes a client session can remain inactive before the server disconnects the session.

The server process determines whether a client has been inactive for a specified period of time, and if so, the server disconnects the client and backs out any related active transactions.

`DefaultFixedFont`

```
DefaultFixedFont=typeface, [ size point-size, keyword, . . . ],
   [ script=scriptname ]
```

Lets you specify the default fixed display font.

typeface

A valid typeface name, such as Times or Courier.

point-size

The size of the font, in points. If you omit the point size, OpenEdge uses the default point size as defined by the typeface.

keyword

You can specify the following keywords: bold, italic, underline, or strikeout.

scriptname

By default, OpenEdge uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, OpenEdge writes the specified script name to the font entry.

DefaultFont

```
DefaultFont=typeface, [ size point-size, keyword, . . . ],
   [ script=scriptname]
```

Lets you specify the default display font.

typeface

A valid typeface name, such as Times or Courier.

point-size

The size of the font, in points. If you omit the point size, OpenEdge uses the default point size as defined by the typeface.

keyword

You can specify the following keywords: bold, italic, underline, or strikeout.

scriptname

By default, OpenEdge uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, OpenEdge writes the specified script name to the font entry.

DefaultV6UpdateFont

```
DefaultV6UpdateFont=typeface, [ size point-size, keyword, . . . ],
   [ script=scriptname]
```

Lets you specify the Version 6 update font.

typeface

A valid typeface name, such as Times or Courier.

point-size

The size of the font, in points. If you omit the point size, OpenEdge uses the default point size as defined by the typeface.

keyword

You can specify the following keywords: bold, italic, underline, or strikeout.

scriptname

By default, OpenEdge uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, OpenEdge writes the specified script name to the font entry.

When V6Display=YES, the font specified by the DefaultV6UpdateFont setting is used when a fill-in is in update mode. The default value for DefaultV6UpdateFont is a version of the system default fixed font with underlines (`fixedsys`, `underline`).

The DefaultFixedFont and the DefaultV6UpdateFont should be the same base font. If you change them, make sure that the font is fixed and that the DefaultV6UpdateFont has the underline attribute. OpenEdge does not automatically add the underline attribute to the DefaultV6UpdateFont setting.

`EndMsgTitle EndMsgText`

```
EndMsgTitle=title-text
EndMsgText=message-text
```

Lets you specify the title and text for the Windows shutdown message box.

When Windows shuts down, a dialog box prompts the user to verify the shutdown. If the user answers `YES`, OpenEdge closes and the Windows Exit procedure continues. If the user answers `NO`, the Windows Exit procedure terminates and Windows remains active.

EndMsgTitle can be up to 80 characters long. Its default value is "OpenEdge." EndMsgText can be up to 80 characters long. Its default value is "Windows is exiting. Is this OK?".

`FrameSpacing`

```
FrameSpacing=n
```

Lets you specify how the OpenEdge layout processor spaces frames.

Setting the value to -1 places the frame at the next character unit boundary. Setting the value to 0 places frames one half of a character unit apart. Setting the value greater than 0 spaces the frames by the specified number of pixels.

ImmediateDisplay

```
ImmediateDisplay={ YES | NO }
```

Lets you redisplay the screen at predefined intervals.

If you do not set this option, some Version 6 applications will not draw the screen correctly. Set this option for applications with existing code that you do not want to modify. The default setting is NO.

You can also use the IMMEDIATE-DISPLAY attribute with the SESSION system handle to surround the offending code. This is the preferred technique.

For more information about the IMMEDIATE-DISPLAY attribute and the SESSION system handle, see *ABL Reference.*.

Keep3DfillinBorder

```
Keep3DfillinBorder={ YES | NO }
```

Set to YES to display a disabled fill-in field with a border. Set to NO to display a disabled fill-in field without a border. The default value is NO.

MultitaskingInterval

```
MultitaskingInterval=n
```

Lets you specify how the OpenEdge session interacts with a Windows cooperative multi-tasking environment. Its value determines how often OpenEdge filters events or messages between itself and other Windows applications. As OpenEdge filters these events more often, it executes procedures less efficiently but allows other Windows applications more opportunity to execute. Adjusting the internal event filter is particularly useful during background processing, such as report generation.

To maximize performance during batch-mode processing, set the value to 9999 (the maximum value allowed). If you want to run another application while you run OpenEdge in batch mode, set the value to 1000. If you set the value to 0 (the default), the internal loop never executes. Although this results in high performance, interoperability with other Windows applications is poor. The lower the number, the more often the loop executes, resulting in better interoperability. Set this option for applications with existing code that you do not want to modify.

You can also use the MULTITASKING-INTERVAL attribute with the SESSION system handle to control how OpenEdge interacts with Windows cooperative multi-tasking by surrounding long ABL processing loops with this attribute.

See *ABL Reference.* for more information about the MULTITASKING-INTERVAL attribute and the SESSION system handle.

NoSplash

```
NoSplash={ YES | NO }
```

Set to YES to turn off display of the splash screen. By default, the OpenEdge GUI client and the WebClient display a splash screen during startup. The screen, which is a blue rectangle with the words "Powered by Progress Software," disappears when the OpenEdge application starts.

NoZGrouping

```
NoZGrouping={ YES | NO }
```

Set to YES to disable the grouping of ABL windows at startup. By default, the AVM groups all ABL windows created in a session such that when an ABL window is brought to the foreground all other ABL windows move to the foreground with it. Grouping is automatically disabled when an application displays a .NET form.

Opsys

```
Opsys=character-string
```

Lets you specify the value used by the OPSYS preprocessor directive or the value returned by the OPSYS function. Valid settings include any character string.

PopupAppOverridesSys

```
PopupAppOverridesSys={ YES | NO }
```

Set to YES to display the system-default pop-up menu only if a user-defined pop-up menu is not available. Set to NO to display the system-default pop-up menu regardless of a user-defined pop-up menu. If a user-defined pop-up menu is available, it is displayed after the system pop-up menu is closed. The default value is YES.

Note: The system-default pop-up menu is displayed when you click the right mouse button on a fill-in field or an editor widget and a user-defined pop-up menu is not available.

PopupNoSysDefault

```
PopupNoSysDefault={ YES | NO }
```

Set to YES to suppress the system-default pop-up menu. Only the user-defined pop-up menu is displayed. The default value is NO.

Note: The system-default pop-up menu is displayed when you click the right mouse button on a fill-in field or an editor widget and a user-defined pop-up menu is not available.

PopupOnMouseDown

```
PopupOnMouseDown={ YES | NO }
```

Set to YES to display a pop-up menu when the right mouse button is depressed. Set to NO to display a pop-up menu when the right mouse button is released. The default value is NO.

Note: This option applies only to pop-up menus that are displayed with the right mouse button (that is, MENU-MOUSE = 3). Menus that are displayed with the left or middle mouse button are always displayed when the buttons are depressed.

PrinterFont

```
PrinterFont=typeface, [ size=screen-point-size]
```

Lets you specify the printer fonts OpenEdge uses with the OUTPUT TO PRINTER statement.

PrinterFont*n*

```
PrinterFontn=typeface, [ size=screen-point-size]
```

Lets you specify the printer font OpenEdge uses with the OUTPUT TO LPT*n* statement.

UseClipChildren

```
UseClipChildren={ YES | NO }
```

This option is set to YES by default and works with the Windows manifest file in drawing buttons. For more information on the manifest file, see Windows manifest file on page 80.

Do **not** change the default setting of the UseClipChildren option until you first talk with a Progress Software Technical Support representative.

UseNative3D

```
UseNative3D={ YES | NO }
```

This option is set to YES by default. It specifies to use Windows 3D drawing capabilities.

Set this option to NO to use OpenEdge's 3D drawing capabilities instead. If you set the UseNative3D option to NO, you **must** also remove the prowin32.exe.manifest file from your *installation-path*\bin\ directory. If you set this option to NO and you do **not** remove the manifest file, you will see inconsistencies in the appearance of your user interface.

Note: The 64-bit client ignores the UseNative3D=NO setting and continues to use Windows 3D drawing capabilities.

UseSourceEditor

```
UseSourceEditor={ YES | NO }
```

Set to YES if you want to use the Source Editor with advanced editing features. Set to NO if you want to use Rich Edit editor. Both editors are Unicode-enabled.

Use-3D-Size

```
Use-3D-Size={ YES | NO }
```

Lets you specify how OpenEdge calculates the height of a character unit.

If you set the value to YES, OpenEdge uses the height of a three-dimensional fill-in. If you set the value to NO, OpenEdge uses the height of a non-three-dimensional fill-in. The default setting is YES.

V6Display

```
V6Display={ YES | NO }
```

Lets you specify Progress Version 6 behavior.

When you compile a procedure in Windows, OpenEdge compiles the borders and fonts associated with user interface widgets into the screen layout by default. As a result, you might encounter compilation problems when you try to run a Version 6 application in a later version of OpenEdge on Windows. There might not be enough space on the display to accommodate the borders and default fonts that are automatically associated with user interface widgets of the Version 6 application.

To avoid this problem, set V6Display to YES to specify that you are running Version 6 code.

You can also set a session attribute to control the run-time setting of the V6Display state. The attribute is V6DISPLAY and has a Boolean value. Setting this attribute has the same effect as specifying the setting at startup; however, you must hide all existing windows.

When you set the V6Display option or session attribute:

- The character unit height is recalculated to the height of the font specified by the DefaultV6UpdateFont option.
- The character unit width is recalculated to the width of an average character in the font specified by the DefaultV6UpdateFont option.

- The default font is automatically changed to use the font specified by the DefaultFixedFont option.
- All OpenEdge fill-ins are created without a border.
- OpenEdge fill-ins keep any trailing spaces.
- All OpenEdge fill-ins that are enabled for update use the font specified by the DefaultV6UpdateFont option (by default, the default fixed font with underlines). This means that updateable fill-ins are drawn with underlines.
- Frame titles are drawn so that the frame title height measures exactly one character unit.

The V6Display option or attribute does not enable support for `PUT SCREEN`. In addition, if you set the V6Display option, OpenEdge applications that have not been compiled with the V6Display option set to `YES` will draw screens incorrectly.

Note: The V6Display option is intended only for use with applications, not with the OpenEdge ADE. The ADE is not supported when V6Display is set to `YES`.

If you plan to use Version 6 user interface code with tools based on later OpenEdge releases, Progress Software Corporation recommends that you use the V6Frame option instead of V6Display.

`V6FontNumber`

```
V6FontNumber=n
```

Lets you specify the font for Version 6 frames.

By default, this option points to the font defined as font3 in the [Fonts] section. Font3 is the ADE character-mode font. If you change the V6FontNumber value, make sure you define the new font number in the [Fonts] section.

Note: Do not change the definition of font3 in the [Fonts] section. OpenEdge relies on that font definition for other purposes.

`V6FrameBGC`

```
V6FrameBGC=n
```

Lets you specify the default background color for all frames that use the `V6FRAME` option. Valid settings include any number that corresponds to a color in the OpenEdge color table.

`WindowSystem`

```
WindowSystem=character-string
```

Lets you specify the value used by the `WINDOW-SYSTEM` preprocessor directive. Valid settings include any character string.

Colors section

The Colors section of the `progress.ini` file contains options that specify the colors that make up the color table for use with graphical user interface clients. Although there are 16 preset colors available to an application for use as either foreground or background colors, you can specify up to 256 colors.

Note: Do not change the first 16 colors. These colors are reserved for the Application Development Environment (ADE) tools. Changing them could cause the OpenEdge ADE tools to malfunction.

Each entry in the Colors section performs two semantic functions: it defines a color, and it maps that color to an integer in the range 0 to 255. An application specifies this integer when making a color assignment to a widget.

Specify a color entry using the following syntax:

Syntax

```
colorn = { R , G , B , | colorname }
```

n

An integer from 0 to 255 that specifies the color table entry.

R

An integer that specifies the amount of red present in the color.

G

An integer that specifies the amount of green present in the color.

B

An integer that specifies the amount of blue present in the color.

colorname

Any of the following color names, mapped to the colors defined in the Windows Control Panel. For example:

```
COLOR-SCROLLBAR
COLOR-BACKGROUND
COLOR-ACTIVECAPTION
COLOR-INACTIVECAPTION
COLOR-INACTIVECAPTIONTEXT
COLOR-MENU
COLOR-WINDOW
COLOR-WINDOWFRAME
COLOR-MENUTEXT
COLOR-WINDOWTEXT
COLOR-CAPTIONTEXT
COLOR-ACTIVEBORDER
COLOR-INACTIVEBORDER
COLOR-APPWORKSPACE
COLOR-HIGHLIGHT
COLOR-HIGHLIGHTTEXT
COLOR-BTNFACE
COLOR-BTNHIGHLIGHT
COLOR-BTNSHADOW
COLOR-GRAYTEXT
COLOR-BTNTEXT
```

Any entries you add to the color table must be sequential. For example, since the installed `progress.ini` file defines color0 to color15, the next color you add must be color16. If you add an entry for color17, OpenEdge ignores it if color16 is undefined.

For backward compatibility, color0 to color15, as installed, are the same as the colors supported in Version 6. See the chapter about colors and fonts in *OpenEdge External Programming Interface Reference* for a listing of these colors.

For backward compatibility, the `progress.ini` file specifies the following color pairs for Version 6 applications:

- `NORMAL` specifies the colors for fill-ins that do not have input focus.
- `INPUT` specifies the colors for fill-ins that have input focus.
- `MESSAGES` specifies the colors for the message area.

`NORMAL`, `INPUT`, and `MESSAGES` each take a pair of integer values from 0 to 255. The integers represent color table entries. For example, in the default `progress.ini` file, `NORMAL` is defined as foreground equal to color table entry 0 (RGB value 0,0,0) and background equal to color table entry 15 (RGB value 255,255,255).

The following example shows how to redefine `NORMAL` as foreground equal to color table entry 3 and background equal to color table entry 12:

```
NORMAL=3,12
```

Default Window section

The Default Window section of the `progress.ini` file allows you to specify:

- The location of the default window as an x,y pixel value.

- The size of the default window as a number of rows and columns. The default size is 25 by 80.

The following is a sample Default Window section:

```
[Default Window]
x=
y=
rows=
columns=
```

Fonts section

The `progress.ini` file as installed contains font table entries for the OpenEdge ADE tools. OpenEdge uses the default system fonts defined in the Startup section, unless you add additional fonts to the existing font table in the Fonts section.

Note: You can add site-specific fonts to the file, but do not change the first 8 fonts. These fonts are reserved for the Application Development Environment (ADE) tools. Changing them could cause the OpenEdge ADE tools to malfunction.

Determine which display fonts exist in your environment. The fonts you choose should be graphically compatible. One way to choose fonts is to use the font common dialog box.

The font table can contain up to 256 entries. Each entry in the table performs two semantic functions: it names a font that exists in your environment and it maps that font to an integer from 0 to 255. An application specifies this integer when making a font assignment to a widget.

Note: You cannot enter extra spaces in the font definition section. Extra spaces will cause the error 4499.

Specify a font entry using the following syntax:

Syntax

```
fontn=typeface, [ size point-size, keyword, . . . ],
   [ script=scriptname]
```

n

An integer that specifies the font table entry.

typeface

A valid typeface name, such as Times or Courier.

point-size

The size of the font, in points. If you omit the point size, OpenEdge uses the default point size as defined by the typeface.

keyword

You can specify the following keywords: bold, italic, underline, or strikeout.

scriptname

By default, OpenEdge uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, OpenEdge writes the specified script name to the font entry.

WinChar Startup section

The WinChar Startup section of the `progress.ini` file contains options that specify the following environment characteristics for character clients:

- Default interface attributes
- Default key bindings

The WinChar Startup section also contains OpenEdge environment variables that you can set for use with character clients. See Specifying environment variables on page 78 for a list of environment variables you can set in the WinChar Startup section.

The WinChar Startup section contains the following options:

`BrowseRowMarker`

```
BrowseRowMarker=n
```

Lets you specify a character to represent a browse row marker. Valid settings include any number that corresponds to a character in the `CPSTREAM` character set.

`CursorHeight`

```
CursorHeight=n
```

Lets you specify the height of a cursor as a percentage. Valid settings include any number between 0 and 100. This option overrides the UseDosCursor setting.

`EnableMouse`

```
EnableMouse={ YES | NO }
```

Set to `YES` to enable mouse input. Set to `NO` to disable mouse input. The default value is `YES`.

FastCursorMode

```
FastCursorMode={ YES | NO }
```

For WebSpeed agents, this option is set to YES by default so that the ABL client does **not** process the normal hiding and showing of the cursor during display. This is the default behavior for WebSpeed agents since WebSpeed applications do not have cursors (they do not have a console). This option setting improves the performance of WebSpeed applications.

For character clients, this option is set to NO by default. Using FastCursorMode=YES with the character client on Windows can cause incorrect cursor behavior so make sure you leave this option set to NO if you are **not** using WebSpeed.

Note: To avoid possible performance issues and incorrect cursor behavior, do **not** change the default setting of this option until you first talk with a Progress Software Technical Support representative.

MenuMnemonicColor

```
MenuMnemonicColor=n
```

Lets you specify the color of mnemonic characters in menu items. Valid settings include any number that corresponds to a color in the OpenEdge color table.

SingleLineBorder

```
SingleLineBorder=n,n,n,n,n,n,n,n
```

Lets you specify the characters used to draw borders and lines (such as frame borders, rectangles, submenu and pop-up menu borders, and horizontal and vertical lines). Valid settings include any eight numbers that correspond to characters in the CPSTREAM character set, separated by commas.

These eight numbers represent the corners and sides of a rectangle and must be specified in the following order: top, bottom, left, right, top-left corner, top-right corner, bottom-right corner, and bottom-left corner.

SysCheckmark

```
SysCheckmark=n
```

Lets you specify a character to represent a selection marker in widgets (such as a selection list, menu, or browser). Valid settings include any number that corresponds to a character in the CPSTREAM character set.

UseDosCursor

```
UseDosCursor={ YES | NO }
```

Set to YES to change the cursor from an underscore to a solid block when switching from insert mode to over-strike mode in a fill-in field or editor widget. Set to NO to use an underscore as the cursor in either mode. The default value is NO.

V6Keys

```
V6Keys={ YES | NO }
```

Set to YES to use a set of key bindings that are Version 6 compatible by default. The default value is NO.

V6FKeys

```
V6FKeys={ YES | NO }
```

Set to YES to use the bindings for the **CTRL**, **ALT**, **SHIFT**, and **F13** to **F39** keys that are Version 6 compatible by default. The default value is NO.

WinChar Colors section

The WinChar Colors section of the progress.ini file contains options that specify the colors that make up the color table for use with character clients.

Each entry in the WinChar Colors section performs two semantic functions: it defines a color, and it maps that color to an entry in the color table by specifying an integer from 0 to 255. An application specifies this integer when making a color assignment to a widget.

Specify a color entry using the following syntax:

```
keyword
              | colorn = { [ BLINK- ] [ BRIGHT- ] fgnd-color | bgndcolor }
```

keyword | colorn

You can also specify color0, color1, and color2 using the NORMAL, INPUT, and MESSAGES keywords, respectively. The keyword overrides the color*n* specification. That is, if you specify both color*n* and its corresponding keyword, OpenEdge displays the colors you specified with the keyword.

fgnd-color

The color to use for the screen foreground.

bgnd-color

The color to use for the screen background.

For information about color definitions for the WinChar Colors section, see the Color Phrase reference entry in *ABL Reference*..

WinChar Default Window section

The WinChar Default Window section of the `progress.ini` file specifies the line mode for a character client. You can specify 50 rows if your hardware supports 50-line mode. Specify 25 rows if you want the application to be compatible with Progress Version 6 or if you want to port the application to a character terminal.

The following is an example of a WinChar Default Window section:

```
[WinChar Default Window]
rows=25
```

WinChar Keys section

The WinChar Keys section of the `progress.ini` file defines the customized key bindings for use with character clients. The following table lists the user-definable key bindings in the Windows environment. For a complete listing of the Windows key function and key label mappings for handling user input, see *OpenEdge External Programming Interface Reference*.

Table 6: User-definable key bindings in Windows

Key function	Key label
APPEND-LINE	CTRL+A
BLOCK	CTRL+V
BREAK-LINE	CTRL+ALT+B
CLEAR	F8
COPY	F11
CUT	F10
DEFAULT-POP-UP	SHIFT+F4
DELETE-LINE	CTRL+D
END-ERROR	ESC
ENTER-MENUBAR	F3
FIND	CTRL+F
GET	F5

Key function	Key label
GO	F1
HELP	F2
INSERT-MODE	INS
NEW-LINE	CTRL+N
NEXT-FRAME	CTRL+TAB
PASTE	F12
PREV-FRAME	CTRL+SHIFT+TAB
PUT	F6
RECALL	F7

Specify a key-binding entry using the following syntax:

Syntax

```
keyfunction=keylabel [ , keylabel]
```

Use a single entry to specify a key binding; additional entries for the same key function are ignored.

If you specify more than one key label, OpenEdge uses the first when it needs to display a key label given a key function, as in "Enter data or press Escape to end."

For example, the following entry maps the Help key function to the **F2** key label, and the Go key function to the **F1** key label:

```
HELP=F2
GO=F1
```

You can use any combination of **CTRL**, **SHIFT**, and **ALT** keys. However, **SHIFT** only has an effect when used with ASCII keys. If an entry contains nonexistent key labels, the entry is ignored.

Note: Set the V6Keys option in the [WinChar Startup] section to enable OpenEdge to use a set of key bindings that are Version 6 compatible by default.

Debug-Init section

Do not modify information in the Debug-Init section.

Debug-Macros section

The Debug-Macros section of the `progress.ini` file contains a list of parameters that define initial macros used by the Debugger. You can also define your own macros. For more information, see *Troubleshoot ABL Applications*.

Debug-Buttons section

The Debug-Buttons section of the `progress.ini` file contains a list of parameters that define initial buttons used by the Debugger. You can also define your own buttons. For more information, see *Troubleshoot ABL Applications*.

Keys section

The Keys section is an optional section you can create in the `progress.ini` file to customize key bindings for the Windows environment.

The following table lists the static key bindings in the Windows environment.

Table 7: Static key bindings in Windows

Key function	Key label
BACKSPACE	BACKSPACE
BACK-TAB	SHIFT+TAB
CURSOR-DOWN	CURSOR+DOWN
CURSOR-LEFT	CURSOR+LEFT
CURSOR-RIGHT	CURSOR+RIGHT
CURSOR-UP	CURSOR+UP
DELETE-CHARACTER	DEL
END	END
HOME	HOME
PAGE-DOWN	PAGE-DOWN
PAGE-UP	PAGE-UP
RETURN	ENTER
TAB	TAB

The following table lists the user-definable key bindings in the Windows environment.

Table 8: User-definable key bindings in Windows

Key function	Key label
APPEND-LINE	No default
BLOCK	No default
BREAK-LINE	No default
CLEAR	No default
DEFAULT-POP-UP	F4
DELETE-LINE	No default
END-ERROR	ESC
FIND	No default
GET	No default
GO	F2
HELP	F1
INSERT-MODE	INS
NEW-LINE	No default
NEXT-FRAME	F6
PREV-FRAME	SHIFT+F6
PUT	No default
RECALL	No default

Keymap section

The Keymap section is an optional section you can create in the `progress.ini` file to define mappings between standard ASCII (7-bit) characters and extended (8-bit) characters. Extended characters are typically non-English alphabetical characters. OpenEdge uses these keymap entries to build a translation table for processing characters in the input and output streams.

Note: Using the Keymap section is the Version 6 technique for mapping characters. Using conversion tables is a better technique for processing characters. For information about defining conversion tables, see *Internationalize ABL Applications*.

Use the `IN` statement, in a keymap entry, to map an ASCII character to an extended character using the following syntax:

Syntax

```
[ MAP-language]
  :IN(\ascii-char)=\extended-char:
```

language

An identifier that appears with the MAP option.

ascii-char

The 7-bit character, in octal (*nnn*) format. You must specify all three digits.

extended-char

The 8-bit character, in octal (*nnn*) format. You must specify all three digits.

Use the OUT statement, in a keymap entry, to map an extended character to an ASCII character using the following syntax:

```
[ MAP-language]
  :OUT(\extended-char)=\ascii-char:
```

You can use multiple IN and OUT statements in a single keymap entry.

If you use extended character sets, you must specify character mappings for each character set on a per-terminal basis. For example, the following code fragment demonstrates how to map characters to German language characters:

```
[MAP-german]
  :IN(\102)=\341:
  :IN(\101)=\216:
  :IN(\141)=\204:
  :IN(\117)=\231:
  :IN(\157)=\224:
  :IN(\125)=\232:
  :IN(\165)=\201:
```

The IN statements in the preceding example map ASCII characters (B, A, a, O, o, U, and u) to German characters (ß, Ä, ä, Ö, ö, Ü, and ü). When OpenEdge sees the character A (\101) on input, it converts the character to Ä (\216). Likewise, when OpenEdge needs to send an Ä to the terminal, it sends (\101). The terminal sees A and displays Ä.

Suppose the terminal cannot display an Ä. You can use the OUT statement to specify an appropriate character to display for Ä, such as capital A. For example:

```
[MAP-german]
  :OUT(\216)=\101:
```

You can also specify keymap entries for the input and output streams using the MAP option with each of the following ABL language elements:

- `INPUT FROM` statement
- `INPUT THROUGH` statement
- `INPUT-OUTPUT THROUGH` statement
- `OUTPUT THROUGH` statement
- `OUTPUT TO` statement

For more information about these language elements, see *Internationalize ABL Applications*.

Specifying environment variables

The following table lists the environment variables you can set in either the Startup section or the WinChar Startup section of the `progress.ini` file.

Table 9: Environment variables

Variable	Description	Code example
PROPATH	A list of directory paths separated by commas. When you run a procedure from within an OpenEdge session with the RUN statement, OpenEdge searches for it in the directory paths defined in PROPATH in the order listed. **Note:** When OpenEdge encounters a period in PROPATH (".,C . . ."), it substitutes the pathname of the current directory.	`PROPATH=.,C:\Progress\OpenEdge`
PROMSGS	The full pathname of the OpenEdge error messages file. The default value is install-dir\PROMSGS. You only have to set the PROMSGS environment variable if you want to use an error messages file different from the default PROMSGS file in the installation directory.	`PROMSGS=C:\Progress\OpenEdge\PROLANG\GER\PROMSGS.GER`
PROCFG	The path name of your product's configuration file. The configuration file is a data file that identifies the OpenEdge product and components that you are licensed to use. Reset PROCFG if you have moved the configuration file from the directory in which you installed OpenEdge.	`PROCFG=C:\Progress\OpenEdge\PROGRESS.CFG`

Variable	Description	Code example
PROSTARTUP	The name of the OpenEdge startup parameter file. The default is `startup.pf`.	`PROSTARTUP=C:\Progress\OpenEdge\startup.pf`
DLC	The pathname of the directory in which you installed the OpenEdge system software.	`DLC=C:\Progress\OpenEdge`

Windows icons for running OpenEdge

The install process places icons in the OpenEdge group on the Windows Explorer desktop and in the Windows Start menu. The default group name is OpenEdge; however, you can choose another group name. The icons vary depending on the products installed. Modifying client icons on page 80 describes the icons.

Table 10: Windows-installed icons

Icon title	Icon command line
OpenEdge Client	*install-dir*\bin\prowin32.exe
OpenEdge Run-time Client	*install-dir*\bin\prowin32.exe -rr
OpenEdge Help	*install-dir*\prohelp*filename*
OpenEdge Desktop	*install-dir*\bin\prowin32.exe -p _desk.p
OpenEdge AppBuilder	*install-dir*\bin\prowin32.exe -p _ab.p

On the icon command line, `install-dir` is the directory where you have installed the products.

See the Windows user documentation for more information about creating additional icons, copying and modifying existing icons, and general information about the Windows Explorer.

Starting the OpenEdge client

You can start the OpenEdge client using any of the following techniques:

- Choose the icon on the desktop.
- Choose **StartRun** from the task bar and enter startup parameters on the command line.
- Click on **prowin32** from the Explorer.

Starting the OpenEdge ABL client places you in the Procedure Editor with no databases attached.

Modifying client icons

You can edit the startup command options, including the environment variables, for the default OpenEdge clients.

To modify startup parameters for a client icon:

1. Select the **client** icon from the OpenEdge program group and choose **FileProperties** from the Windows Explorer, or right-click on the icon and choose **Properties** from the pop-up menu. The **Properties** dialog box appears.
2. From the **Shortcut** tab, modify the command-line in the **Target** field. You can supply up to 259 characters in this field. The command-line limit of 259 characters is still valid for the expanded version of the command-line text.
3. Choose **OK**.

Note: You can specify environment variables in the command-line. For example, specifying the following command runs the `PROWIN32` executable located in the path where the `%DLC%` environment variable points:

```
%DLC%\BIN\PROWIN32.EXE
```

Windows manifest file

The OpenEdge installation includes a *manifest file*—a text file that contains XML code. The manifest file provides your ABL application with the look of the Windows XP user interface. For OpenEdge, the manifest file is named `prowin32.exe.manifest` and it is located in the *`install-dir`*`/bin` directory by default. For Web Client, the manifest file is `prowc.exe.manifest`. (Each executable needs its own manifest file and the manifest file must be in the same directory as the executable for which it is intended.)

Note: This Windows XP feature also applies to Windows Vista and Windows 7.

Only Windows XP recognizes manifest files; all other versions of Windows (including Windows XP with the screen display set to the Windows Classic style), ignore the manifest file. If you do not want to use the manifest file on XP, you must move, rename, or delete it.

The following settings in the `[Startup]` section of your `progress.ini` file work in conjunction with the manifest file:

- **`UseNative3D`** — When you set this option to `NO`, you specify to use OpenEdge's 3D drawing capabilities insteadof Windows 3D drawing functionality. If you set the UseNative3D option to `NO`, you **must** also remove the `prowin32.exe.manifest` file from your *`installation-path`*`\bin\` directory. If you set this option to `NO` and you do **not** remove the manifest file, you will see inconsistencies in the appearance of your user interface.

Note: The 64-bit client ignores the `UseNative3D=NO` setting and continues to use Windows 3D drawing capabilities.

- **`UseClipChilden`** — This option works with the Windows manifest file in drawing buttons. The default setting for this option is `YES`. Do **not** change the default setting of the UseClipChildren option until you first talk with a Progress Software Technical Support representative.

- **`ButtonImageBorderMode`** — This setting lets you change the default behavior of how OpenEdge draws borders on buttons with images. One of the factors that determines the effect of this option is the presence of a manifest file.

See Startup section on page 58 for more information on these settings.

Environments with OpenEdge GUI for .NET applications

OpenEdge GUI for .NET supports the ability to access Microsoft .NET objects from ABL much as if they were native ABL classes. The GUI for .NET provides an alternative user interface for ABL applications based on .NET forms and controls. GUI for .NET applications have extra considerations, including the following:

- Microsoft .NET Framework 3.0 or later
- .NET assemblies
- Code Access Security (CAS) for network installs

.NET Framework

Any client machine running the GUI for .NET requires .NET Framework 3.0 or later. The OpenEdge installer includes the option to install .NET Framework 3.0 if it is missing. Choose this option if you are going to develop or deploy GUI for .NET applications on a client machine. The installer determines if there is already an appropriate version of the framework installed before installing the version included in OpenEdge.

Note: Microsoft requires users to accept an EULA before installing the .NET Framework. The Microsoft EULA appears during the OpenEdge installation if the framework is installed. This prevents silent installs on client machines that do not already have an appropriate version of the framework.

By default, the AVM only loads the .NET Common Language Runtime (CLR) or .NET assemblies as the OpenEdge tools or a running GUI for .NET application require them. This behavior avoids wasting resources running unnecessary code. If your session definitely requires .NET, consider using the Preload CLR (`-preloadCLR`) startup parameter to force the CLR and any specified assemblies to load at startup. For more information on `-preloadCLR`, see *Startup Command and Parameter Reference*.

Running a GUI for .NET application on a client machine without the .NET Framework produces the following error:

```
The OpenEdge GUI for .NET is disabled for this session; the required .NET Framework
version is not installed (14798).
```

.NET assemblies

An *assembly* is a .NET library or collection of classes. A GUI for .NET application uses several assemblies to connect the ABL and .NET sides of the application. The application might also use assemblies for the following:

- Optional OpenEdge Ultra Controls for .NET
- Classes used by the application that reference .NET objects
- Third-party .NET controls

The AVM internally knows how to locate some assemblies that the GUI for .NET requires. By default, the AVM looks for any other assemblies in the session's working directory. The working directory must contain an Assembly References File (`assemblies.xml`). This file lists all the assemblies used by the application. Progress Developer Studio for OpenEdge includes a mechanism for creating an `assemblies.xml` file for any project that uses the GUI for .NET. Progress Developer Studio for OpenEdge automatically adds assembly references to the `assemblies.xml` file, but it also allows you to manage those references more directly. You can use the Assembly References Tool (`proasmref`) for modifying the `assemblies.xml` file outside of Progress Developer Studio for OpenEdge. You can add any additional assemblies that your application uses to that file or create your own.

Note: Even if you install assemblies in .NET's Global Assembly Cache (GAC), you must list all assemblies used by your application in `assemblies.xml`.

If you do not put your assemblies and `assemblies.xml` file in the session's working directory, you must use the Assemblies (`-assemblies`) startup parameter. The Assemblies (`-assemblies`) startup parameter enables you to specify the absolute or relative path to the directory containing `assemblies.xml` and any third-party assemblies that are not installed in the GAC. A relative path is relative to the working directory for the OpenEdge session. For more information on `-assemblies`, see *Startup Command and Parameter Reference*.

In exceptional circumstances, you might need to include an `assemblies.config` file in your assemblies directory. The `assemblies.config` file is a customized type of .NET application configuration file. It applies only to assemblies that your application expects to find in the assemblies directory. It cannot affect assemblies installed in the GAC or the internal assemblies in *`OpenEdge-install-dir`*`/bin`.

The `assemblies.config` file enables the CLR to find assemblies by providing assembly-specific information. When a GUI for .NET application loads a .NET assembly, OpenEdge checks this configuration file for any changes for that assembly. For example, the file can point to an updated version of an assembly, an alternate path to load an assembly, or provide localization support by pointing to alternate assemblies. For more information on application configuration files, see the "Configuration Files" topic in the *.NET Framework Developer's Guide*, currently at the following address:

```
http://msdn.microsoft.com/en-us/library/1xtk877y(VS.80).aspx
```

Code Access Security for network installs

The .NET Framework includes a system of Code Access Security (CAS) that tries to prevent untrusted code from performing privileged operations. When the CLR loads an assembly at runtime, it uses CAS to determine whether the assembly has permission to perform the requested operations. By default, the framework gives assemblies installed on a network share limited permissions. These limited permissions would prevent the GUI for .NET from running.

OpenEdge's Shared Network Installation Utility provides an option to adjust these permissions to allow the GUI for .NET to run. You should consult with your IT administrator about any security issues before using this option. For more information, see the section on shared network installations in *Install OpenEdge*.

Maintaining the UNIX user environment

Typically, you create a user-defined startup script to specify environment variable settings in the UNIX software environment. You can also use the `buildenv` script. OpenEdge uses the `buildenv` script to maintain environment variables that define objects, libraries, and options required for building executables on UNIX.

You must also maintain the PROTERMCAP file. PROTERMCAP is a terminal definition file that contains the terminal description for your monitor's terminal type. OpenEdge uses this file to determine how to interact with your terminal, console, or terminal emulator.

Maintaining the buildenv script

The buildenv script contains settings for the following types of options:

- Options required by OpenEdge
- Options required by OpenEdge tools
- User-defined options
- Environment variables

The following shows an excerpt from a sample buildenv script.

Table 11: Sample buildenv script on UNIX

```
#!/bin/sh
MACHINE="aix"
export MACHINE
UDORA="User Defined"
export UDORA
if [ x$RDL = x -a -f `pwd`/buildenv.log ] ; then
    rm -f `pwd`/buildenv.log
fi
ORA_MK=$DLC/oebuild/make/ORA.mk
if [ x$RDL != x ]; then
    ORA_MK=$PRGSSRC/project/ORA.mk
fi
# If ORALIB is not set already, attempt to set it
# ORACLE_HOME must be set for the following to work.  Error if unset.
if [ "x$ORALIB" = x -a "x$ORACLE_HOME" != x ] ; then
    # Execute ('source' into current environment) oralib.sh if it exists.
    # (it is assumed to be a Bourne-shell compatible script which, when
    # finished, has defined an appropriate value for $ORALIB.
    if [ -x $DLC/oebuild/build/oralib.sh ] ; then
        . $DLC/oebuild/build/oralib.sh
    else . . .
```

You can edit this script to customize environments at different sites. The buildenv script is located in the $DLC/oebuild/make directory. OpenEdge uses the buildenv script to maintain environment variables that define objects, libraries, and options required for building executables on UNIX. For more information about building OpenEdge executables, see Building OpenEdge Executables on page 181

Maintaining the PROTERMCAP file

The terminal definition file for most character interfaces is called the PROTERMCAP file. This file is a flat file database that contains the terminal description for your monitor's terminal type to inform OpenEdge on how to interact with your terminal, console, or terminal emulator. It specifies which characters to use to draw boxes, how to write in various video modes (such as inverse or blinking), how many lines are on the screen, and what actions to perform for each key on the keyboard.

Note: Make a backup copy of the PROTERMCAP file before making modifications.

Each terminal entry contains the commands and terminal descriptions that OpenEdge uses to communicate with and control the terminal. Each entry is divided into these logical sections:

- Terminal name
- Terminal capabilities
- OpenEdge terminal-specific key functions
- OpenEdge color table
- Vermont Views key functions
- Pointer to key functions

The PROTERMCAP environment variable (*install-dir*/protermcap): points to the PROTERMCAP file.

The PROTERMCAP file contains a lengthy entry for each terminal type supported by OpenEdge. Each entry has six parts. The following shows a partial example of each part of a PROTERMCAP entry.

Parts of a PROTERMCAP entry

```
wy370|wyse370|wyse 370 running in native mode:\        Terminal name
  :  :\
  :ct:\
  :cm=\E[%i%d;%dH:\              Terminal capabilities
  :co#80:\
  :li#24:\
  :G1=k:
  :  :\
  :GO(F1)=\EOP:\                OpenEdge terminal-specific
  :HELP(F2)=\EOQ:\              key functions
  :  :\
  :COLOR 5 RED/BLACK=\E[31;40m:\E[m:\            OpenEdge
  :COLOR 6 GREEN/BLACK=\E[32;40m:\E[m:           color table
  :  :\
  :ku=\E[A:  :L_ku=<Up>:\
  :kd=\E[B:  :L_kd=<Down>:\         Vermont Views
  :kr=\E[C:  :L_kr=<Right>:\        key functions
  :kl=\E[D:  :L_kl=<Left>:\
  :kh=\E[E:  L_kh=<Home>:\
  :tc=v7kf:                     Pointer to Version 7 key functions
```

Note: There is a 8,192-byte limit on the size of each PROTERMCAP entry.

The following sections describe PROTERMCAP syntax and each part of a PROTERMCAP entry.

PROTERMCAP syntax

This section explains the following:

- The general syntax rules that apply when specifying fields in any of the six PROTERMCAP sections.
- The syntax rules that apply when specifying string values within a field. String values can occur in the terminal capabilities section, the key bindings section, and the color table section.

Syntax rules that apply only to specific sections of the PROTERMCAP file are explained in context.

General syntax

A PROTERMCAP entry is a single logical line that contains many fields. The syntax for an entry is as follows:

- Each field is terminated by a colon (:). (Use the octal equivalent \072 to specify a colon that is not a terminator.)
- Unless stated otherwise, a field cannot contain embedded separators. Spaces and tabs improve readability, but they are considered empty fields and must be terminated by a colon.
- Comment lines begin with a pound sign (#), and can only occur at the beginning and end of each `PROTERMCAP` terminal entry (that is, before the terminal name line and after the pointer to key functions line).
- To enhance readability, the `PROTERMCAP` file, as installed, contains as many physical lines as there are fields. Each physical line in the entry contains the following, in the order listed:
  - An empty field consisting of a tab character
  - A field definition
  - A backslash (\)
  - A return

  The backslash indicates that the terminal entry continues on the next physical line of the file. If you need to include a literal backslash in a field, use two backslashes.

The following code fragment from the entry for the Wyse 370 shows the use of the colon, tab, space, backslash, and return characters:

```
wy370|wyse370|wyse_370|wyse 370 running in native mode:\
  :  :\
  :ct:\
  :cm=\E[%i%d;%dH:\
  :co#80:\
  :li#24:\
  :G1=k:
  :  :\
  :GO(F1)=\EOP:\
  :HELP(F2)=\EOQ:\
  :  :\
  :COLOR 5 RED/BLACK=\E[31;40m:\E[m:\
  :COLOR 6 GREEN/BLACK=\E[32;40m:\E[m:
```

String syntax

Certain fields in the capabilities, key bindings, and color sections are specified using strings. Strings are typically a combination of command sequences, literal values, and encoded arguments that tell OpenEdge how to control some aspect of the display.

The following table show the symbols that are common to all strings. Symbols that are unique to strings specified in the capabilities, key bindings, or color sections are explained in context.

Table 12: Syntax for specifying string values

Mnemonic	Description
\E	ESC (ASCII 27)
\n	New line
\r	Return

Mnemonic	Description
\t	Tab
\b	Backspace
\f	Form feed
999	Octal representation of a character. A colon (:) within a string must be represented as \072 and a null character as \200.
^*x*	**CTRL+x**, where *x* represents an appropriate character. For example, ^N represents **CTRL+N**.
\^	^
\\	\

Some terminal types require you to specify a delay period, also called a padding constant, with certain string capabilities. The delay, in milliseconds, gives the terminal adequate time to execute the command. Specify this delay as an integer value after the equals sign (=) in a string, as shown in the following example:

```
:cm=20\E[%i%d;%dH:\
```

This tells OpenEdge to wait 20 milliseconds after sending the cursor motion (cm) command sequence.

You can optionally follow an integer delay value with an asterisk (*) for commands that affect more than one line of the display. In this case, OpenEdge delays the specified time for each line affected by the command. However, this significantly degrades performance and usually is not required.

When you specify proportional delay, you must specify an integer as the delay in a string field. It cannot have a fractional part; for example, a delay of 1.5* is not valid.

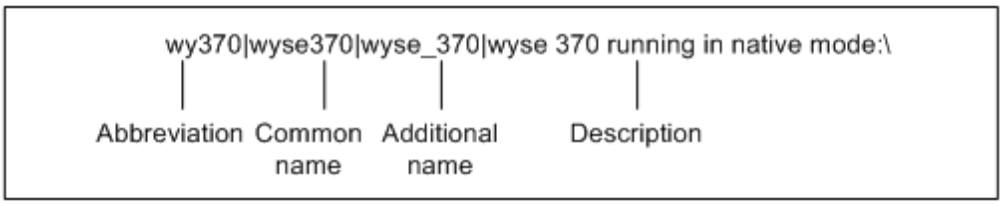
Note: Do not specify a padding constant with the GS, GH, GV, GE, G1, G2, G3, and G4 capabilities.

Terminal name entries

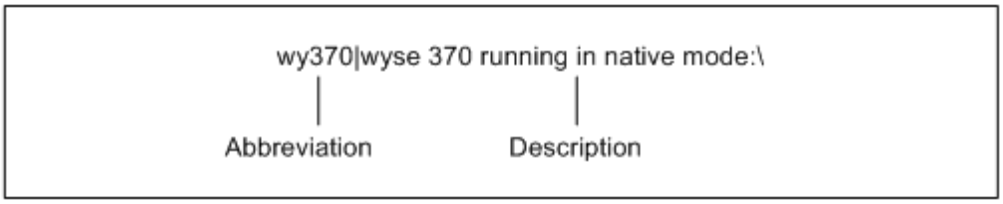
The first field in each PROTERMCAP terminal entry is the Name field. This field provides the names known for the terminal. The Name field contains one or more of the following components, separated by the pipe symbol (|):

- An abbreviation of the terminal's name
- The common name of the terminal
- Additional names for the terminal, if any
- A description of the terminal in which only the last name can contain blanks for readability (optional)

PROTERMCAP requires at least one of the name entries (1, 2, or 3, above) to be entered without spaces; the other two, and the description, are optional. See the following example:



Another example of a terminal name entry is:



Terminal capabilities entries

The terminal capabilities section of a terminal entry contains fields that tell OpenEdge features of the terminal being used. These entries are indented from the name line to distinguish them from the terminal name.

Terminal capabilities entries control visual attributes of the terminal: how big the terminal screen is, how to move to any point on the screen, how to refresh the screen, how to enter/exit special display modes (reverse, blinking, underline), etc. They also control specific terminal functions to set up and clean up the terminal properly, so as not to leave a terminal in an unexpected state (which might leave the terminal unusable).

These entries are the core sequences that OpenEdge uses to communicate with the terminal. Changing them can cause very unpredictable results and might even cause OpenEdge to put the terminal in an unusable state. The entries can be changed, but exercise great care in doing so.

Terminal capability entries consist of:

- A two-character mnemonic that identifies a generic terminal capability
- An assignment operator equal sign (=), or a pound sign (#)
- For certain capability types, a value that allows OpenEdge to understand and interact with the capability on a specific terminal

For example:

```
:ct:\
:cm=\E[%i%d;%dH:\
:co#80:\
:li#24:\
:G1=k:\
```

Functional listing of terminal capabilities

These are the cursor terminal capabilities that OpenEdge supports:

bc, CF, CN, cm, kd, kl, kr, ku, ke, ks

These are the color/video terminal capabilities that OpenEdge supports:

BB, BR, ct, HS, HR, se, so, ue, us

The following are the graphics character terminal capabilities that OpenEdge supports:

G1, G2, G3, G4, GH, GV, GE, GS

Note: Do not use a padding constant with any of the above graphic strings.

The following are miscellaneous terminal capabilities that OpenEdge supports:

co, li, CA, ce, cl, is, Se, Si, te, ti, pc, xi

The following are the key translation capabilities that OpenEdge supports:

k0, k1, k2, k3, k4, k5, k6, k7, k8, k9, k0, k. (period), k, (comma), k- (hyphen)

The following are the scroll region capabilities that OpenEdge supports:

al, sf, sr, dl, cs

Note: The ce, cl, and cm capabilities are required for OpenEdge to start up. Vermont Views supports all the capabilities listed here, except ce, te, and ti, and the key translation capabilities.

The following table describes the mnemonics that OpenEdge supports.

Table 13: Alphabetical listing of capability mnemonics

Mnemonic	Data type	Description
al	STRING	Add line.
bc	STRING	Backspace character.
BB	STRING	Enter COLOR 4 mode (usually blink mode).
BR	STRING	Exit COLOR 4 mode (usually blink mode).
CA	BOOLEAN	Terminal clears screen with current attribute. If you do not specify and this capability is available, full-screen clears with attributes other than NORMAL will be slow.
ce	STRING	Clear to end of line.
CF	STRING	Cursor off.
cl	STRING	Clear screen. This improves speed. On some terminals, you must define cl as home cursor followed by clear to end of screen.
cm	STRING	Cursor motion.
CN	STRING	Cursor on.
co	INTEGER	Columns on screen (usually 80).
cs	STRING	Change scroll region. Uses cm syntax.
ct	BOOLEAN	Terminal supports color.

Mnemonic	Data type	Description
dl	STRING	Delete line.
G1	STRING	Single-line upper-right corner character.
G2	STRING	Single-line upper-left corner character.
G3	STRING	Single-line lower-left corner character.
G4	STRING	Single-line lower-right corner character.
GE	STRING	graphics end. Sent when finished drawing box or underline.
GH	STRING	Single-line horizontal character.
GS	STRING	graphics start. Sent to begin drawing box or underline.
GV	STRING	Vertical-line graphic character.
HR	STRING	Exit COLOR 3 mode (usually highlight mode).
HS	STRING	Enter COLOR 3 mode (usually highlight mode). This is usually set to high intensity, if available.
is	STRING	Terminal initialization string. Sent when OpenEdge starts.
k0-9 k. k, k-	STRING	Define the codes sent by the numeric keypad keys if these are different from the codes sent by the standard 0-9, period, comma, and hyphen keys. These are only used by OpenEdge.
kd	STRING	Down arrow.
ke	STRING	Exit keypad mode.
kl	STRING	Left arrow.
kr	STRING	Right arrow.
ks	STRING	Set keypad mode.
ku	STRING	Up arrow.
li	INTEGER	Lines on screen (usually 24).
pc	CHARACTER	Pad character (defaults to null).
Se	STRING	String to send when OpenEdge terminates (after te).
se	STRING	Exits COLOR 2 mode, or MESSAGE color (usually reverse video).

Mnemonic	Data type	Description
sf	STRING	Scroll forward. You can use this on terminals that have scrolling regions (for example, terminals with cs and sr defined).
Si	STRING	String to send when OpenEdge starts (after is).
so	STRING	Enter COLOR 2 mode, or MESSAGE color (usually reverse video). OpenEdge uses this attribute by default for the two-line message area at the bottom of the screen. It is also the default for the PROMPT-FOR color of selectable widgets (that is, buttons, sliders, toggle-boxes, etc.). If not set and the PROMPT-FOR color of a widget is not explicitly set in ABL, a widget might not be visible.
sr	STRING	Scroll reverse.
te	STRING	Cursor movement string to send when OpenEdge terminates.
ti	STRING	Cursor movement string to send when OpenEdge starts.
ue	STRING	Exit COLOR 1 mode, or INPUT color.
us	STRING	Enter COLOR 1 mode, or INPUT color (usually the underline attribute). OpenEdge uses COLOR 1 as the default PROMPT-FOR color for fill-ins and editors. If COLOR 1 is not defined by the us and ue capabilities, and the PROMPT-FOR color for fill-ins and editor widgets is not explicitly set in ABL, these widgets might not be visible, and there might be no indication they are enabled.
ws	BOOLEAN	Uses the device /dev /tty for the current port to determine the number of columns and rows for the terminal. Meaningful primarily for emulated terminals in a windowing system (that is, xterm). If successful, overrides co and li; otherwise, defaults to co and li. OpenEdge will not automatically resize if the size of the terminal changes after initialization.
xi	BOOLEAN	Terminal will not automatically do a hardware scroll when the last position (last row and last column) on the screen is written to. Specify whether available, otherwise OpenEdge will not write to the last position in order to avoid the automatic scrolling.

There are four types of terminal capability fields: BOOLEAN, NUMERIC, CHARACTER, and STRING. The following table summarizes the four data types and the operators used in conjunction with each. Boolean fields are specified by a mnemonic abbreviation only. The mnemonic indicates that the feature is present and its absence indicates that the feature is not present. All other capabilities take a value, which can be a string, a character, or a numeric value.

Table 14: Data types and operators

Data type	Operator
BOOLEAN	none
NUMERIC	# and value

Data type	Operator
CHARACTER	= and character
STRING	= and string

The following code fragment from the terminal entry for the Wyse 370 terminal shows how to specify BOOLEAN, NUMERIC, CHARACTER, and STRING capabilities:

```
:ct:\
:cm=\E[%i%d;%dH:\
:co#80:\
:li#24:\
:G1=k:\
```

In this example, the mnemonic ct is a Boolean switch that indicates that the terminal supports color. The mnemonic cm specifies cursor motion and takes a string. The assignment operator for string values is an equals sign (=). The mnemonics co and li specify the number of columns and lines on the display and each takes an integer. The assignment operator for integer values is a pound sign (#). The mnemonic G1 specifies the upper-right corner line-graphic character and takes a character. The assignment operator for character values is an equals sign (=).

Vermont Views key function capabilities

Certain programs, such as the install program, use a character interface library called Vermont Views for character input. Vermont Views does not use the OpenEdge Key Function syntax of keyfunction(label)=sequence. Instead, it uses key function and key label capabilities. Except for bc, ku, kd, kl, and kr, these key function capabilities are unique to Vermont Views. The key label capabilities mnemonics are created from the key function capabilities mnemonics by prepending "L_". Because both OpenEdge and Vermont Views use only the first instance of a capability, each key function capability can represent only one key label.

For readability, the Vermont Views section of an entry has two distinct columns containing a list of the:

1. Key function capabilities.
2. Matching key label capabilities. The label in this column always starts with "L_".

   Here is an example:

```
:ku=\E[A:           :L_ku=<Up>:\
:kd=\E[B:           :L_kd=<Down>:\
:kr=\E[C:           :L_kr=<Right>:\
:kl=\E[D:           :L_kl=<Left>:\
:kh=\E[E:           :L_kh=<Home>:\
Vermont Views                   Labels
mnemonics
```

Note: For readability in the PROTERMCAP file, the bc, ku, kd, kl, and kr capabilities are usually repeated in the Vermont Views part of the file, even though both OpenEdge and Vermont Views use these capabilities as they appear earlier in the OpenEdge capabilities part. To avoid confusion, the key sequences for the Vermont Views capabilities should be the same as the OpenEdge capabilities.

The following table summarizes the Vermont Views key function capabilities mnemonics.

Table 15: Vermont Views key function mnemonics

Mnemonic	Description
kh	Home (beginning of line, then beginning of page)
EN	End (end of line, then end of page)
PU	Page up
PD	Page down
ki	Insert toggle
DL	Delete character
ESC	Exit current form
bt	Back tab
fk0	Function Key 1
fk1	Function Key 2 (contents)
fk2	Function Key 3 (enter the menu bar, then exit the menu bar)
fk3	Function Key 4 (exit current form)
fk4	Function Key 5 (browse backward)
fk5	Function Key 6 (browse forward)
fk6	Function Key 7 (go back through hypertext links)
fk7	Function Key 8 (bring up search dialog box)
fk8	Function Key 9
fk9	Function Key 10
Aka	Browse backward
Akd	Browse forward
Aki	Exit current form
Akp	Go back through hypertext links
Aks	Bring up search dialog box

Mnemonic	Description
Aku	Menu bar
Akw	Contents

Pointer to key functions

The last line of every terminal entry is a pointer to the OpenEdge key functions. These key functions are common to all terminal types and are used by OpenEdge to map certain key functions with certain control sequences, for example, **`:FIND (CTRL-F) = ^F:`**. This pointer always appears as follows:

```
tc=v7kf:
```

If you modify or create new `PROTERMCAP` entries, be sure to include this line as the last line in the entry. This line ensures that the terminal entry can support current OpenEdge releases.

If you want to use Version 6 mappings with later OpenEdge releases, change the pointer as follows:

```
tc=v6kf:
```

Specifying colors

You specify colors in the OpenEdge terminal-specific key functions and capabilities section of `PROTERMCAP`. You can specify up to 123 color fields in a terminal entry. This manual refers to the color specifications as the color table.

Each line in the table performs two semantic functions: it defines a foreground/background color pair and it maps that color pair to an integer from 5 to 127. The integer is how an application references the color pair when making color assignments to a widget. (An application typically assigns two color pairs to a widget: one for display mode and another for prompt mode.)

Use the following syntax to specify a color field in a `PROTERMCAP` terminal entry:

Syntax

```
COLOR color-number [color-name] =start-sequence:stop-sequence:
```

color-number

An integer from 5 to 127 that specifies the location of the color in the color table. The color number is the mechanism used to assign a color pair in an application.

color-name

The name of the color. You can use any color value except an ABL keyword. Although this is optional, including a name makes the color section of the terminal entry self-documenting (you cannot embed comments) and also makes the color specifications backwardly compatible.

start-sequence

The character sequence that starts the color attribute.

stop-sequence

The character sequence that stops the color attribute.

For example, the following fields from the Wyse 370 terminal define color table locations 5 and 6:

```
COLOR 5 RED/BLACK=\E[31;40m:\E[m:\
COLOR 6 GREEN/BLACK=\E[32;40m:\E[m:\
```

OpenEdge reserves color table locations 0 to 4 as follows:

- Color 0 holds the Version 6 color `NORMAL`, which is set by the terminal initialization (is) field.
- Color 1 holds the Version 6 color `INPUT` and is set and cleared by us and ue. As installed, this is underline mode.
- Color 2 holds the Version 6 color `MESSAGE` and is set and cleared by so and se. As installed, this is reverse-video.
- Color 3 holds the colors used for high intensity and is set and cleared by HS and HR.
- Color 4 holds the colors used for blink and is set and cleared by BB and BR.

Note: As installed, the `PROTERMCAP` file does not support spacetaking (embedded attribute) terminals. These are terminals that use characters on the display to hold control codes for video and color capabilities. If you add support for such terminals, OpenEdge ignores the capability when displaying the interface.

Specifying the cursor motion capability

The absolute cursor-motion (cm) capability allows OpenEdge to move to any row and column on the display. The capability takes a string that is composed of two parts:

- Control characters
- Two conversion specifications

The conversion specifications allow OpenEdge to convert an integer row value and an integer column value into the arguments the terminal expects.

The cm string syntax is exactly the same as that used in the standard UNIX `termcap` file, in which the conversion specifications follow the pattern of the UNIX `printf( )` command. The following table lists the supported conversion specifications, each of which begins with a percent sign (`%`).

Table 16: Syntax for specifying cursor motion string value (*value*)

Mnemonic	Description
`%d`	Send the integer to the terminal in ASCII format.
`%2`	Pads the value with a leading blank if it is only one digit.

Mnemonic	Description
%3	Pads the value with up to two leading blanks.
%.	Treats the value as an ASCII character value.
%+x	Sets value += x, then does a %
%>xy	If value **> x**, then value +=y; no output
%r	Reverses the order of row and column; no output
%i	Allows row and column values based on a zero origin to be incremented by one to make them appropriate for terminals whose origin is at (1,1)
%%	Gives a single **%**
%n	Exclusive OR row and column with 0140
%B	BCD (16*(value/10)) + (value%10); no output
%D	Reverses coding (value - 2*(value%16)); no output
%x	Gives a two-byte, zero-padded, hex value as in the "%0.2x" format for the UNIX printf() function
%M	Not similar to any UNIX printf( ) format, and given the row (row) and column (col), %Mx%Mx translates to:**p1 = row / x + x p2 = row mod x + x p3 = col / x + x p4 = col mod x + 2x**Where p1 through p4 are sent to the terminal as characters

The following example is the cursor motion field for the wyse 370 terminal:

```
:cm=\E[%i%d;%dH:\
```

Where the following symbols indicate the specified action:

- **\E[** — The standard ANSI console lead-in sequence (Escape, followed by a left square bracket). OpenEdge transmits this literally.
- **%i** — Instructs OpenEdge to add one to the row and column values to compensate for a terminal whose origin is 1,1.
- **%d** — Instructs OpenEdge to send the row and column values to the terminal in ASCII format.
- **;** — Separates the row value from the column value.
- **H** — A terminating character.

To create a cm string for a new terminal type, find the description of this capability in your terminal's documentation to determine the control characters and the arguments it expects to move the cursor to a random spot on the display. The documentation specifies an algorithm for converting a row/column integer pair to the arguments it needs. The manual also specifies the order in which the row and column values are expected (typically row first), and the character to use to separate the two values. For example, to move to column 65, row 66, a terminal might expect the characters A and B, which have ASCII values of 65 and 66, respectively.

Specifying keyboard mappings

The PROTERMCAP file maps key functions to key labels. The following table shows the mappings as installed. Many of the mappings shown are terminal specific, and certain terminals might not support the function for a particular sequence given in the table.

Table 17: OpenEdge PROTERMCAP key functions

Key function	UNIX key label
ABORT	CTRL-\ CTRL-ALT-DEL
APPEND-LINE	CTRL-A
BACKSPACE	BACKSPACE CTRL-H DEL-CHAR
BACK-TAB	BACK-TAB CTRL-U SHIFT-TAB
BELL	BELL CTRL-G
BLOCK	SELECT CTRL-V
BOTTOM-COLUMN	ESC-CTRL-B
BREAK-LINE	ESC-B
CANCEL-PICK	ESC-CTRL-X
CHOICES	ESC-CTRL-H
CLEAR	F8 CTRL-Z
CLOSE	ESC-Z
COMPILE	ESC-P
COPY	F11 ESC-C
CURSOR-DOWN	CURSOR-DOWN CTRL-J
CURSOR-LEFT	CURSOR-LEFT CTRL-O
CURSOR-RIGHT	CURSOR-RIGHT CTRL-L
CURSOR-UP	CURSOR-UP CTRL-K
CUT	F10 ESC-X
DEFAULT-POP-UP	ESC-U
DELETE-CHARACTER	DEL
DELETE-COLUMN	ESC-CTRL-Z

Key function	UNIX key label
DELETE-END-LINE	ESC-K
DELETE-FIELD	ESC-CTRL-D
DELETE-LINE	REMOVE CTRL-D
DELETE-WORD	ESC-D
EDITOR-BACKTAB	CTRL-B
EDITOR-TAB	CTRL-G
END	END ESC -> ESC-.
END-ERROR	F4 PF4 ESC CTRL-E
ENTER-MENUBAR	F3 PF3 ESC-M
EXIT	ESC-Q
FIND	CTRL-F FIND
FIND-NEXT	ESC-F
FIND-PREVIOUS	ESC-I
GET	F5 ESC-O
GO	F1 PF1 CTRL-X DO
GOTO	ESC-G
HELP	HELP F2 ESC-? PF2
HOME	HOME ESC-H ESC-h ESC-< ESC-,
INSERT-COLUMN	ESC-CTRL-N
INSERT-FIELD	ESC-CTRL-G
INSERT-FIELD-DATA	ESC-CTRL-F
INSERT-FIELD-LABEL	ESC-CTRL-E
INSERT-MODE	INS F9 CTRL-T CTRL-@
LEFT-END	ESC-LEFT-ARROW ALT-LEFT-ARROW
MAIN-MENU	ESC-CTRL-M
MOVE	ESC-CTRL-V

Key function	UNIX key label
NEW	ESC-N
NEW-LINE	INSERT-HERE CTRL-N
NEXT-ERROR	ESC-E
NEXT-FRAME	ESC-CTRL-I
NEXT-WORD	CTRL-W
OPEN-LINE-ABOVE	ESC-L
OPTIONS	ESC-CTRL-O
PAGE-DOWN	PAGE-DOWN ESC-CURSOR-DOWN ESC-DOWN-ARROW NEXT-PAGE NEXT-SCRN
PAGE-LEFT	ESC-W
PAGE-RIGHT	ESC-Y ESC-CTRL-J
PAGE-RIGHT-TEXT	ESC-CTRL-J
PAGE-UP	PAGE-UP ESC-CURSOR-UP ESC-UP-ARROW PREV-PAGE PREV-SCRN
PASTE	F12 ESC-V
PICK	ESC-CTRL-P
PICK-AREA	ESC-CTRL-W
PICK-BOTH	ESC-CTRL-Q
PREV-FRAME	ESC-CTRL-U
PREV-WORD	CTRL-P
PUT	F6 ESC-S
RECALL	F7 CTRL-R
REPLACE	ESC-R
REPORTS	ESC-CTRL-A
RESUME-DISPLAY	CTRL-Q
RETURN	RETURN CTRL-M

Key function	UNIX key label
RIGHT-END	ESC-CURSOR-RIGHT ESC-RIGHT-ARROW ALT-RIGHT-ARROW
SAVE-AS	ESC-A
SCROLL-LEFT	ESC-CTRL-L
SCROLL-MODE	ESC-T
SCROLL-RIGHT	ESC-CTRL-R
SETTINGS	ESC-CTRL-@
STOP	CTRL-C CTRL-BREAK
STOP-DISPLAY	CTRL-S
TAB	TAB CTRL-I
TOP-COLUMN	ESC-CTRL-T

Using key function syntax

To change any of these mappings or create new ones, use the following syntax:

Syntax

```
keyfunction(key-label) = sequence:
```

keyfunction

The name of a key function.

key-label

The key label as it appears on the keyboard.

sequence

The characters transmitted when the key is pressed.

As in other sections of the PROTERMCAP file, string values are assigned using an equal sign (=) and the field is terminated with a colon (:). For example:

```
RECALL(F7)=^AF\r:
```

This field in a `PROTERMCAP` terminal entry defines the **F7** key as transmitting a **CTRL+A** followed by a capital **F** followed by a carriage return, and associates use of **F7** with the `RECALL` function.

The following field defines **SCROLL-DOWN** to act like **PAGE-DOWN**:

```
PAGE-DOWN(SCROLL-DOWN)=\021:
```

If you assign the same key label to two or more different key functions, you get a warning message when you start OpenEdge. For example, "You cannot use DELETE for both DELETE-CHARACTER and BACKSPACE."

If you use a key in the ABL `ON` statement or `GO-ON` option only, and do not have to assign it to a standard action, then use the following syntax in the `PROTERMCAP` entry:

```
(key-label)=sequence:
```

For example, the following entry indicates that pressing **F16** sends a capital O followed by a carriage return:

```
(F16)=O\r:
```

If any of the control code sequences sent when you press a key on the keyboard begin with a control key, you cannot use that control key on your keyboard and the key does not have its normal OpenEdge meaning. For example, if you specify **CTRL+F** in a control code sequence when creating a key mapping, you can no longer use **CTRL+F** for **FIND**. You have to map another key to the **FIND** action.

The key labels that the UNIX `stty` command specifies for **FLUSH** and **SUSPEND** override the `PROTERMCAP` file's use of the same key labels. For example, if the `stty` settings for **FLUSH** and **SUSPEND** are **CTRL+Q** and **CTRL+S**, you cannot map these key labels to key functions in the `PROTERMCAP` file. If you do, you receive no warning; the labels assume their `stty` meanings at run time and OpenEdge ignores them.

UNIX stty control functions

The installed `PROTERMCAP` file does not map the ABL functions `ABORT`, `STOP`, and `UNIX-END` to key labels. OpenEdge instead uses the key labels that the UNIX `stty` command specifies for **QUIT**, **INTR** and **EOF** respectively.

For example, the following `stty` command specifies that OpenEdge should use **CTRL+** for `ABORT`, **CTRL+C** for `STOP`, and **CTRL+D** for `UNIX-END`:

```
stty quit ^\ intr ^C eof ^D
```

The labels specified by the `stty` command are of two forms: either **CTRL+X** or **DEL** if the **DELETE** key (octal 177, decimal 127) is used. When entering the `stty` command, indicate the control character by holding down the **CTRL** key and pressing the specified key; you do not type a caret (^) followed by the key.

If an entry in the `PROTERMCAP` file uses one of the key labels specified in the `stty` command, you get a warning message when you start OpenEdge. For example, if the `stty` command specifies the **DELETE** key for the `STOP` function and the `PROTERMCAP` file specifies the **DELETE** key for the `DELETE-CHARACTER` function, you receive a warning message.

In UNIX environments that do not use the Bourne shell (for example, the Korn shell or C shell), job control allows you to end a job currently executing on a terminal. In most environments this is initiated using **CTRL+Z**; however, OpenEdge uses this character sequence to clear the editor.

Extended character support entries

You can define mappings between standard ASCII (7-bit) characters and extended (8-bit) characters in the PROTERMCAP file. Extended characters are typically non-English alphabetical characters. OpenEdge uses these mappings to build a translation table for processing characters in the input and output streams.

If you use extended character sets, you must specify character mappings for each character set on a per-terminal basis. For example, the following code fragment from the PROTERMCAP file demonstrates how to map characters on the Wyse 60 keyboard to German language characters:

```
ger|german|wy60 in german mode:\
  :IN(\102)=\341:\
  :IN(\101)=\216:\
  :IN(\141)=\204:\
  :IN(\117)=\231:\
  :IN(\157)=\224:\
  :IN(\125)=\232:\
  :IN(\165)=\201:
```

The IN statements in the preceding example map ASCII characters (B, A, a, O, o, U, and u) to German characters (ß, Ä, ä, Ö, ö, Ü, and ü). When OpenEdge sees the character A (\101) on input, it converts the character to Ä (\216). Likewise, when OpenEdge needs to send an Ä to the terminal, it sends (\101). The terminal sees A and displays Ä.

Suppose the terminal cannot display an Ä. You can use the OUT statement to specify an appropriate character to display for Ä, such as capital A. For example:

```
:OUT(\216)=\101:
```

The IN and OUT statements express both the ASCII character and the extended character in octal (*nnn*) format.

Given the appropriate PROTERMCAP file entries for a language and terminal, set the TERM environment variable to *terminal/language* (for example, TERM=wy60/german).

You can also specify character mappings for the input and output streams using the MAP option with each of the following ABL language elements:

- INPUT FROM statement
- INPUT THROUGH statement
- INPUT-OUTPUT THROUGH statement
- OUTPUT THROUGH statement
- OUTPUT TO statement

For more information about these language elements, see *ABL Reference*.

Creating a PROTERMCAP entry

On UNIX, if OpenEdge does not support your terminal or you need to modify the PROTERMCAP entry for your terminal, refer to the programming documentation from your terminal's manufacturer. It should specify the appropriate commands, values, and key sequences that OpenEdge needs to interact with your terminal.

If you need to create a new PROTERMCAP entry, first check the /etc/termcap file for an entry for your terminal. The /etc/termcap file is the standard terminal definition file used by several UNIX system programs, such as the vi editor. This file is usually provided with UNIX.

If this file contains an entry for your terminal, copy this entry to the PROTERMCAP file. Use this file as a starting point to build an entry for your terminal in the PROTERMCAP file.

If your version of UNIX uses a personal termcap called terminfo, you must modify the terminfo keywords to match the PROTERMCAP capability mnemonics described in the preceding sections.

Testing terminal entries

If you left any required specifications out of the PROTERMCAP file, you will receive an error message when you run OpenEdge. Once in the Procedure Editor, try some operations, such as insert line, delete line, insert character, delete character, and scroll up/down. Also, you should run procedures that require data entry and display messages in the message area.

To fully test the terminal:

1. Open the Procedure Editor.
2. Test to see that the key-functions and key-labels work.
3. Test to see that colors are working correctly on widgets and frames.
4. Test the output to terminal paged to test scrolling capabilities.

Setting up the terminal

Use your terminal's setup mode to synchronize the terminal with the color and key code definitions in its PROTERMCAP entry.

OpenEdge supports terminals based on definitions of their characteristics stored in a file called the PROTERMCAP file. The PROTERMCAP file supplied with OpenEdge has built-in definitions for many terminals. If your terminal is not included in the PROTERMCAP file supplied with OpenEdge, you must make modifications to that file. OpenEdge uses the shell variable or logical PROTERMCAP to find the PROTERMCAP file. By default, OpenEdge looks for this file whenever you start the software in a character environment.

Setting the terminal type

Use the appropriate command to make a new terminal type the current terminal type. The commands assume that the name of the new terminal type is my_term_type.

Table 18: Setting the terminal type

Operating system	Command	Environment
UNIX (Bourne shell)	`TERM="my_term_type"; export TERM`	In `.profile` or at the shell prompt
UNIX (C shell)	`setenv TERM "my_term_type"`	In `.profile` or at the shell prompt

Alternately, you can change the terminal type from within an OpenEdge program with the following ABL statement:

```
TERMINAL = termid
```

See the `TERMINAL` function reference entry in *ABL Reference* for more information.

Note: Terminal names are case sensitive.

Setting the PROTERMCAP environment variable

If you create a new `PROTERMCAP` file, store it in the current working directory. Then set the `PROTERMCAP` environment variable, using the appropriate command from the following table, to allow OpenEdge to use the new file.

Table 19: Setting the PROTERMCAP environment variable

Operating system	Command
UNIX (Bourne shell)	`PROTERMCAP=protermcap; export PROTERMCAP`
UNIX (C shell)	`setenv PROTERMCAP protermcap`

Reverting to the default PROTERMCAP file

Use the appropriate command in the following table at the operating system prompt to go back to using the default `PROTERMCAP` file.

Table 20: Reverting to the default PROTERMCAP file

Operating system	Command
UNIX (Bourne shell)	`unset PROTERMCAP`
UNIX (C shell)	`unsetenv PROTERMCAP`

Copying an existing terminal entry

Use the `tc` (terminal copy) command when you are defining a new terminal entry that is similar to an existing entry. The command appends the specifications for the similar terminal to the new entry, so you do not have to retype the similar information.

The `tc` command takes a string, as shown in the following example:

```
:tc=terminal-name
```

Follow these guidelines when using the `tc` command:

- To override a capability in the copied entry, specify the capability in the new terminal's entry. Commands specified first always override those specified later.
- To suppress the inclusion of a capability that you do not want to specify in the new terminal's entry, specify only the mnemonic, followed by an at symbol (@) in the new terminal's entry. This tells OpenEdge to ignore the capability when it reads it from the similar terminal.
- You cannot use the Version 6 `PROTERMCAP` in later OpenEdge releases. You can edit the `PROTERMCAP` to use the Version 6 key bindings by editing the following line at the end of the file:

```
:tc=v7kf {change to} :tc=v6kf
```

- Make `tc` the last field in the terminal entry.
- Place the similar terminal entry in the `PROTERMCAP` file before entries that reference it.
- The combined length of all the definitions must be less than 8,192 bytes.

For example, the following code fragment is a terminal entry for a Wyse 370 in 132 column mode. The entry specifies the name of the terminal, a terminal initialization sequence, the number of columns, and then uses `tc` to append the specifications for the Wyse 370 terminal, as shown:

```
wy370-132:Wyse 370 in 132 column mode:\
  :is=\E[?3h\E[90;1"p\E(B\E)0\E[63;0w:\
  :co#132:
  :tc=wy370:
```

5

Managing Client Performance

On client systems, OpenEdge® consumes system resources to execute ABL (Advanced Business Language) procedures, store variables and record buffers, sort records, and store temporary files. When managing client performance, be aware of key performance factors such as procedure loading and execution, temporary file I/O, and sorting I/O. In addition to these factors, application design and coding can significantly affect client performance. For information about this topic, see *OpenEdge External Programming Interface Reference*.

For information about database performance, see *Manage the OpenEdge Database* and *Startup Command and Parameter Reference*.

For details, see the following topics:

- Procedure loading and execution
- Distributing OpenEdge files on a network
- Temporary file I/O
- Sorting I/O

Procedure loading and execution

R-code is the intermediate binary code that ABL generates when it compiles ABL source files. This is the code that OpenEdge actually runs when it executes a procedure or class, whether from session compiles or from permanently generated r-code (`.r`) files. This section describes r-code and the performance options related to it. For information about r-code features and functions, see R-code Features and Functions on page 189.

Using procedure libraries to improve r-code performance

You can organize and store r-code in an ABL procedure library. The r-code files in a procedure library are known as *members*. Procedure libraries allow you to manage and execute r-code more efficiently.

OpenEdge provides two types of procedure libraries: standard and memory-mapped. A standard library contains r-code that executes in local memory. A memory-mapped library contains r-code that executes in shared memory. You create procedure libraries by using the PROLIB utility. For information about using the PROLIB utility to create and manage procedure libraries, see Managing Procedure Libraries on page 115.

When loading and executing r-code from operating system files in a directory, OpenEdge must open and close each file individually. When loading and executing r-code from standard or memory-mapped libraries, OpenEdge opens only one file—the library itself—to access all of the members in the library.

When you execute members from either a standard library in local memory or a memory-mapped library in shared memory, you gain the following advantages:

- Faster access to r-code
- Fewer file open and close operations
- Less I/O to temporary files

When you execute members from a memory-mapped library in shared memory, you gain the following additional advantages:

- Even faster access to r-code
- Requires less memory because multiple clients access the same r-code segments in shared memory

R-code execution environment

OpenEdge provides a dynamic environment for loading and executing r-code from operating system files, standard procedure libraries, and memory-mapped procedure libraries. This execution environment consists of the following components:

- **Execution buffer** — The portion of local memory that OpenEdge allocates and uses to store the chain of loaded r-code segments for all standard r-code executing in local memory.
- **R-code swap file (`.rcd`)** — A file that OpenEdge uses to dynamically swap standard r-code segments in and out of the execution buffer.
- **Shared memory buffer** — The portion of shared memory that the operating system allocates and uses to store and execute the r-code segments for all memory-mapped members.
- **Segment descriptor table** — An in-memory table that references the r-code segments required by all executing r-code, including the location of each r-code segment in local or shared memory and usage count information.
- **R-code directory** — An in-memory table that contains information about all r-code executing in local or shared memory, including r-code size, usage count, segment descriptions, and a reference to the segment descriptor table for each segment.

For more information about the r-code execution environment, see R-code Features and Functions on page 189.

OpenEdge loads and executes r-code in different ways, depending on whether you access the r-code file directly or store the r-code in either a standard or memory-mapped procedure library. The following sections provide an overview of how OpenEdge loads and executes r-code from each of these sources.

Loading and executing r-code from a file

When loading r-code from an operating system file in a directory, OpenEdge opens the r-code file, loads the required r-code segments into local memory, and closes the file.

When executing r-code from an operating system file, OpenEdge accesses and executes the r-code segments in the execution buffer in local memory. OpenEdge also swaps segments to and from the r-code swap file, as necessary.

Loading and executing r-code from a standard procedure library

When loading r-code from a standard library, OpenEdge opens the library file and loads the required r-code segments into local memory. The library remains open until the end of your OpenEdge session or until you remove the library from the `PROPATH`. When OpenEdge needs to load an r-code segment for a member in that library, it accesses the open library in local memory. This process is much faster than loading r-code from an operating system file.

When executing r-code from a standard procedure library, OpenEdge accesses and executes the r-code segments in the execution buffer in local memory. OpenEdge does not swap r-code segments to the r-code swap file unless you specify the PROLIB Swap (-pls) startup parameter. By default, OpenEdge reloads segments from the open library in local memory.

Loading and executing r-code from a memory-mapped procedure library

When loading r-code from a memory-mapped library, OpenEdge opens the library and maps it in shared memory where multiple users can access it. The library remains mapped until the end of your OpenEdge session or until you remove the library from the `PROPATH`. When OpenEdge needs to execute an r-code segment for a member in that library, it accesses the library in shared memory. This is even faster than loading r-code from a standard procedure library.

Tuning r-code execution

The following table lists the parameters you use to tune the r-code execution environment for a session.

Table 21: Startup parameters for tuning the execution environment

Startup parameter	Suggested use
Maximum Memory (-mmax)	Use this parameter to set the initial execution buffer ceiling; the default is 3096K. OpenEdge increases the execution buffer dynamically as required, but only after attempting to swap segments to the r-code swap file. Increase this parameter to minimize swapping I/O.
Directory Size (-D)	Use this parameter to set the initial number of r-code directory entries. OpenEdge can reuse directory entries for inactive r-code files. However, reusing an entry requires some overhead. To reduce this overhead, increase the value of the -D parameter. The default is 100 entries, the minimum is 5 entries, and the maximum is 2,147,483,647 entries.

Startup parameter	Suggested use
Stack size (-s)	When you load data definitions for very large tables or use recursive procedures, you might receive an error message directing you to increase this parameter. This parameter changes the size of the stack (an internal memory area used by OpenEdge program modules).
Nested Blocks (-nb)	If memory is severely limited, use this parameter to limit the maximum number of nested procedure blocks allowed.

For more information about these startup parameters, see *Startup Command and Parameter Reference*.

Tuning standard procedure libraries

The following table lists the startup parameters you use to tune standard procedure libraries for a session.

Table 22: Startup parameters for tuning procedure libraries

Startup parameter	Suggested use
PROLIB Memory (-plm)	If you have a large standard library or a memory shortage, use this parameter to allocate a 512-byte cache for the library's directory instead of loading the entire directory into memory. This parameter slows the speed of library access but increases available memory.
PROLIB Swap (-pls)	Use this parameter to allow swapping from a standard library to a r-code swap file. OpenEdge typically reads library-stored r-code from the library instead of swapping to a r-code swap file. However, when using a library over a network, swapping to a r-code swap file might be faster than accessing the remote library.

If you specify the PROLIB Memory (`-plm`) and PROLIB Swap (`-pls`) startup parameters with memory-mapped libraries, OpenEdge ignores them.

For more information about these startup parameters, see *Startup Command and Parameter Reference*.

Monitoring r-code activity

You can monitor execution environment activity by using the startup parameters listed in the following table to collect usage statistics during an OpenEdge client session.

Table 23: Startup parameters for monitoring r-code activity

Startup parameter	Description
Statistics (-y)	Collects procedure access and usage statistics throughout the OpenEdge session and writes them to an output file (by default, `client.mon` in your current working directory). Use the SHOW-STATS statement at any time during the session to instruct OpenEdge to write the statistics immediately.

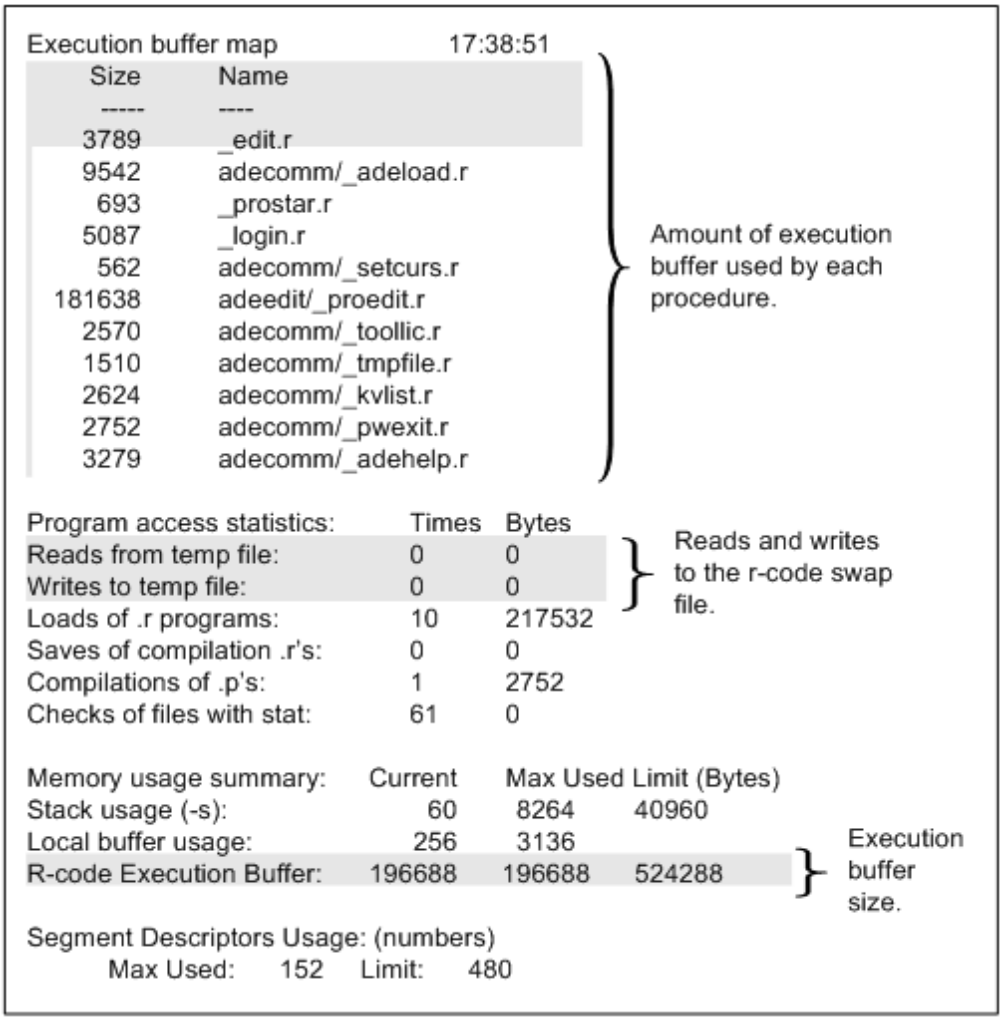
Startup parameter	Description
Statistics with CTRL+C (-yc)	Collects the same statistics as the -y parameter, but lets you press **CTRL+C** as an alternative to executing the SHOW-STATS statement. Use this parameter if you cannot execute SHOW-STATS.
Segment Statistics (-yd)	Collects r-code segment statistics and writes them to the client monitor file (by default, client.mon in your current working directory). It provides a breakdown of an r-code file by segment type, including the number of segments and its size. It also summarizes read and write access to the r-code swap file by segment type and access type, including the number of times the RCD file was accessed and the number of bytes read from or written to the RCD file.
Statistics with Cross-reference (-yx)	Collects procedure call statistics and writes them to an output file (by default, proc.mon in your current working directory). Use this parameter to answer these questions: • How many calls were made in a given period of time? • How long did a procedure spend executing? • How often was a procedure swapped to and from the RCD file?

For more information about these startup parameters, see *Startup Command and Parameter Reference*.

Interpreting r-code usage statistics

The following image shows a sample excerpt from the output of the Statistics (`-y`) startup parameter.

Figure 3: Sample statistics (-y) output



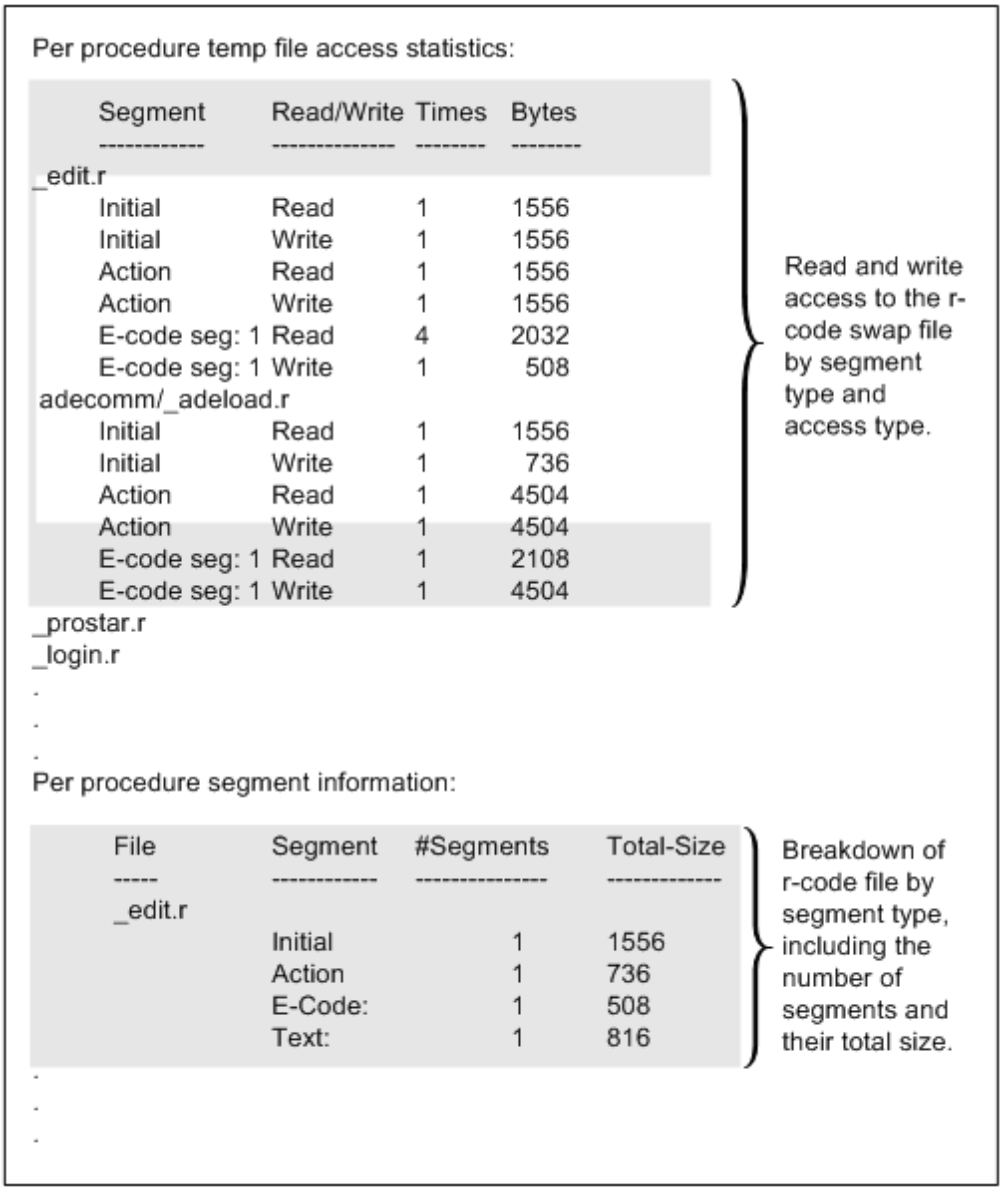
The Reads from temp file and Writes to temp file fields show the amount of swapping to and from the r-code swap file. Segments are written to the swap file only once. Once swapped, they remain in the swap file, even when they are reloaded into memory. For this reason, the number of writes is typically much smaller than the number of reads. If these numbers are relatively high, consider putting the swap file on a dedicated disk. For more information, see Temporary file I/O on page 113.

The R-code Execution Buffer field shows the current size of the execution buffer, the peak usage during the session, and the current ceiling. Use the data in these fields to determine the optimal initial ceiling for the execution buffer (that is, the point at which OpenEdge starts swapping to the r-code swap file). Set this value with the Maximum Memory (`-mmax`) startup parameter. For more information, see Tuning r-code execution on page 107.

Interpreting r-code segment statistics

The following image shows a sample excerpt from the output of the Segment Statistics (`-yd`) startup parameter.

Figure 4: Sample segment statistics (-yd) output



The Per procedure temp file access statistics fields show the swap rates for each segment of each procedure executed during the session. The Per procedure segment information fields show the number and size of r-code segments in each procedure. Use this information to determine how your application design is affecting r-code performance. For more information about application design and r-code performance, see R-code Features and Functions on page 189.

Distributing OpenEdge files on a network

The performance of OpenEdge on your network depends largely on your hardware configuration and where you place the following files:

- OpenEdge system files
- Procedure and class files
- Temporary files
- Log files

Distributing OpenEdge system files

OpenEdge system software files are the executable and support files that run OpenEdge. The OpenEdge installation procedure places these files in the default installation directory on the nodes where you install OpenEdge. If you install OpenEdge on a network file server, you can sometimes improve performance by moving the appropriate system files to the node where they are executed (application workstation or database server machine). This is often true on TCP/IP networks running high-speed UNIX workstations and file servers.

Note: There are security considerations if you intend to use an OpenEdge GUI for .NET application from a network file server. For more information, see Code Access Security for network installs on page 82.

Make sure you copy the correct OpenEdge executables for your LAN protocol. Also, make sure that the operating system on each node knows the new location of the files you copy. Set the appropriate environment variables to point to that directory.

For operating systems with a PATH environment variable or other standard search list convention (such as search directories set with the `map` command in NetWare), be sure that the local OpenEdge installation directory appears before the file server OpenEdge installation directory in the search order. You can do this in the startup file for each network node that receives the OpenEdge system files.

Distributing application files

Application files are files that contain ABL source code (`.p`, `.cls`, `.w`, `.i`) or r-code (`.r`). Store application files in those directories that OpenEdge can access most rapidly, preferably on a local hard disk attached to each application workstation or OpenEdge AppServer machine.

Once you have distributed your application files, modify the `PROPATH` environment variable in the startup files on each application workstation to include the directory search path that contains those files. The following example sets the `PROPATH` environment variable to the working directory (.) and the \ACCTPAY directory on the C: drive of a Windows workstation:

```
SET PROPATH=.;C:\ACCTPAY
```

In Windows, you can use the registry or `progress.ini` file to set this environment variable. For more information, see Maintaining User Environments on page 51.

Distributing temporary files

Temporary files are the files that OpenEdge creates during a session and deletes at the end of a session. Because local I/O is usually much faster than network I/O, store temporary files on the local hard disk or a RAM disk on each application workstation, whenever possible. By default, OpenEdge stores temporary files in the user's working directory, whether the working directory is local or remote. To ensure that OpenEdge stores temporary files in a local directory, either start each client in a local directory or with the Temporary Directory (`-T`) startup parameter set to a local directory.

For more information about temporary files, see Temporary file I/O on page 113. For more information about the Temporary Directory (-T) startup parameter, see *Startup Command and Parameter Reference*.

Distributing log files

OpenEdge creates the following log files:

- **Event log** — Contains a history of significant database events, such as OpenEdge startup and shutdown and system errors. This file always has a `.lg` extension. Place the event log file in the same directory as the database file.
- **Transaction log** — (if you are using the two-phase commit feature) Contains a history of two-phase commit transactions, and uses a `.tl` extension. The transaction log file belongs on the same disk as the database.
- **OpenEdge AppServer log** — (if you are using the OpenEdge AppServer option) Contains OpenEdge AppServer broker and server startup and shutdown information, and it contains other system messages that are not directed elsewhere with the `OUTPUT TO KEEP-MESSAGES` statement. The default location of the AppServer log is the directory where the application broker starts.

Temporary file I/O

OpenEdge creates temporary files during each client session. Each operating system has different temporary files. The following table describes the temporary files.

Table 24: Temporary client session files

File	Description
`.dbi`	Stores temporary tables
`.lbi`	Local before-image file (subtransaction undo)
`.ped`	Edit buffer contents
`.rcd`	Cache of r-code being run in a session
`.srt`	Temporary sort space; session compile storage
`.trp`	Stores Data Dictionary changes until they are saved

By default, OpenEdge stores the temporary files in the user's working directory. Use the Temporary Directory (-T) startup parameter to specify an alternate location. When a session ends, OpenEdge deletes these files unless you specify the Save Temp Files (-t) startup parameter. For more information about the Temporary Directory (-T) and Save Temp Files (-t) startup parameters, see *Startup Command and Parameter Reference*.

On UNIX, unless you use the Save Temp Files (-t) startup parameter, you do not see the temporary files in the directory because they are created unlinked. However, if your system crashes, the UNIX file system recovery program, `fsck`, finds the files. This program might then prompt you to delete the files. If this occurs, delete them.

On Windows, these temporary files are always visible during an OpenEdge session, but the file sizes are set to zero. Windows does not record file sizes until files are closed.

Writing to these temporary files creates I/O activity that might slow down client performance. The amount of temporary file I/O activity varies among applications.

You might find that running a particular procedure results in OpenEdge abnormally terminating in a way that indicates the hard disk is full. However, when you check the disk immediately afterward, adequate space is available. One possibility is that temporary files created during an OpenEdge session become quite large, taking up disk space as they expand. Since OpenEdge erases temporary files at the end of a session, they do not appear when you check the disk. Of the temporary files, the following files are most likely to cause this problem:

- **Temp table file (.dbi)** — OpenEdge uses this file to store temporary tables. If the application uses a lot of temporary tables, this file can grow quite large.
- **Local before-image file (`.lbi`)** — OpenEdge uses this file to back out of subtransactions and variable value changes. If the local before-image file becomes too large, the procedures probably use a lot of subtransactions. Examine the transaction structure of the application procedures to understand why OpenEdge starts so many subtransactions. For more information about transaction processing, see *Develop ABL Applications*.
- **R-code swap file (`.rcd`)** — OpenEdge uses this file to dynamically swap r-code segments in and out of the execution buffer. Use the r-code usage statistics startup parameters to collect information about I/O to the swap file, as explained in Monitoring r-code activity on page 108. Using procedure libraries helps reduce the amount of I/O required to load r-code files. For more information, see Using procedure libraries to improve r-code performance on page 106.

Note: The AVM swaps codepage-converted text segments for procedure libraries from the execution buffer to the r-code swap file and reloads the converted text segments from the swap file instead of the library when necessary.

If temporary files grow to exceed disk space, change the application to reduce the size of temporary files, or use the Temporary Directory (-T) parameter to store the temporary files on a disk with sufficient space to hold the files. To optimize performance, place the temporary files on a dedicated disk.

Sorting I/O

The following table lists the startup parameters you use to improve performance when sorting records for reports and during index rebuild operations.

Table 25: Startup parameters for tuning sort performance

Startup parameter	Description
Speed Sort (-TB)	Increase this parameter to 8K, 16K, or even 31K, particularly for index rebuild operations. Increasing this parameter increases memory usage during sorting and index rebuilds. It also slightly increases disk space usage.
Merge Number (-TM)	Increase this parameter to 32 to speed the merge phase of the sort process (at the cost of increased temporary memory usage).

For more information about these startup parameters, see *Startup Command and Parameter Reference*.

6

Managing Procedure Libraries

An ABL (Advanced Business Language) procedure library is a collection of r-code combined in a single file. The r-code files in a procedure library are known as *members*. It is similar to an object library in Windows or an archive file on UNIX. However, object libraries and archive files are used solely for efficient storage. Procedure libraries allow you to manage and execute r-code more efficiently.

Note: OpenEdge can only make use of r-code and images within a procedure library. It is technically possible to include other types of files in a procedure library. However, the user would need to extract them from the library before making use of them.

For details, see the following topics:

- Library overview
- Setting up a library
- Running members from a library
- Libraries and PROPATH
- Memory and network considerations
- Using the PROLIB utility
- Generating a memory-mapped library
- Code page compatibility

Library overview

OpenEdge provides two types of procedure libraries: standard and memory-mapped. A standard library contains r-code that executes in local memory. A memory-mapped library contains r-code that executes in shared memory. The r-code files that you place in a library are called *members*.

Note: Whether the source code is an ABL procedure or ABL class, it compiles into r-code. For the purposes of this topic, procedure and class files are equivalent.

When you execute a member from either a standard library in local memory or a memory-mapped library in shared memory, you gain the following advantages:

- Faster access to r-code
- Fewer file open and close operations
- Less I/O to temporary files

When you execute a member from a memory-mapped library in shared memory, you gain the following additional advantages:

- Even faster access to r-code
- Requires less memory because multiple users access the same r-code segments in shared memory

When loading and executing r-code from operating system files in a directory, OpenEdge must open and close each file individually. When loading and executing r-code from standard or memory-mapped libraries, OpenEdge opens only one file—the library itself—to access all of the members in the library.

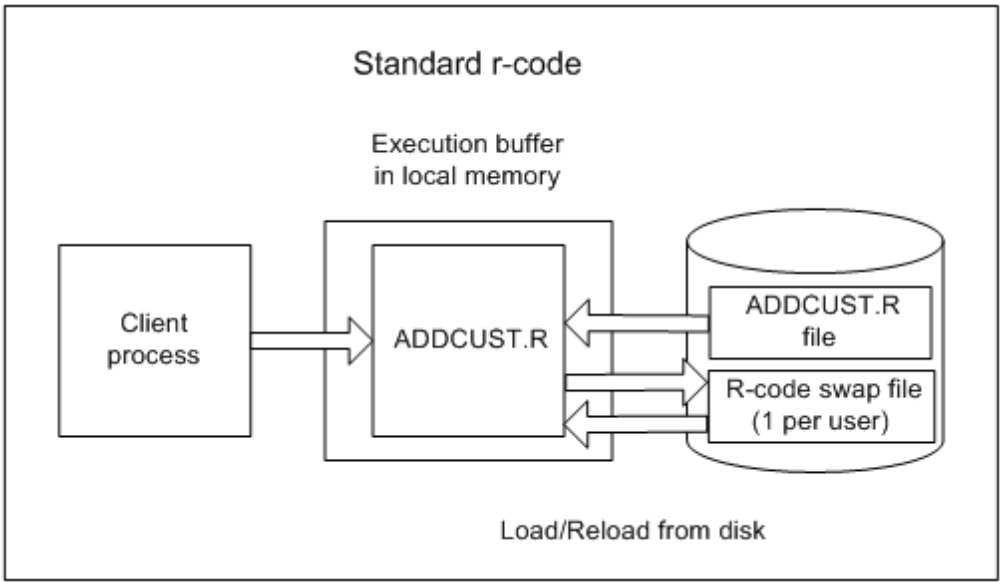
OpenEdge opens a library the first time you run a member from the library. The library stays open until the end of your OpenEdge session or until you remove the library from the PROPATH. For more information about how standard and memory-mapped libraries interact with PROPATH during an OpenEdge session, see Libraries and PROPATH on page 120.

For information about monitoring and optimizing r-code execution during a client session, see ABL Client Deployment Overview on page 21. For more information about r-code structure and execution, see R-code Features and Functions on page 189.

Loading r-code from a file

The following image shows the load operation when executing r-code from an operating system file.

Figure 5: Loading r-code from a file



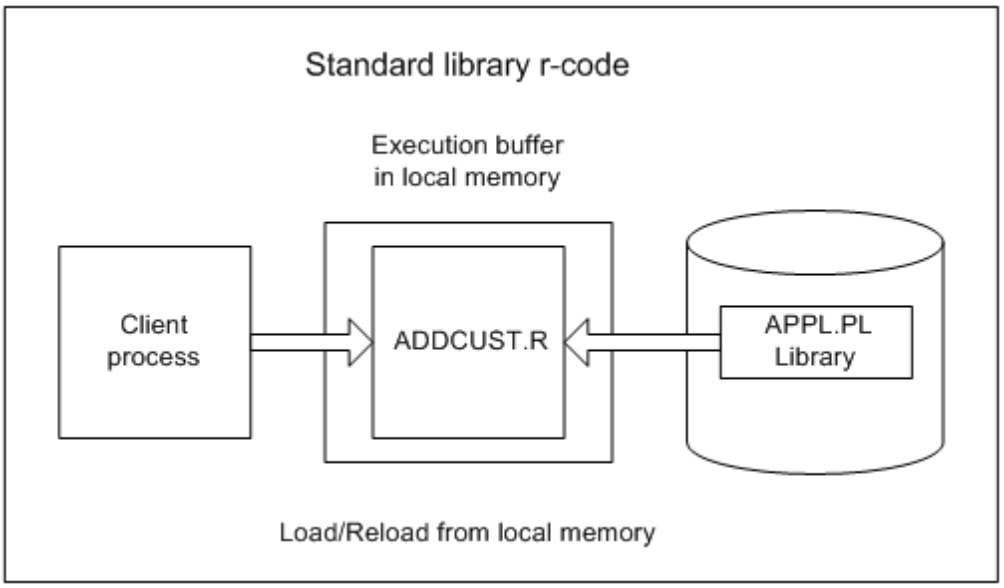
As shown in the above figure, when a client process accesses r-code from an operating system file, OpenEdge loads the r-code into local memory for that individual client and executes the segments from local memory. OpenEdge swaps segments to the r-code swap file, and reloads segments from the r-code swap file, as necessary.

Loading r-code from a standard library

After locating r-code in a standard library, OpenEdge loads the member into local memory.

The following figure shows the load operation when executing r-code from a standard library.

Figure 6: Loading r-code from a standard library



As shown in the above figure, when a client process accesses a member from a standard library, OpenEdge loads the member into local memory for that individual client and executes the segments from local memory. OpenEdge does not swap segments to and from the r-code swap file unless you specify the PROLIB Swap (`-pls`) startup parameter. By default, OpenEdge reloads segments from the open library in local memory.

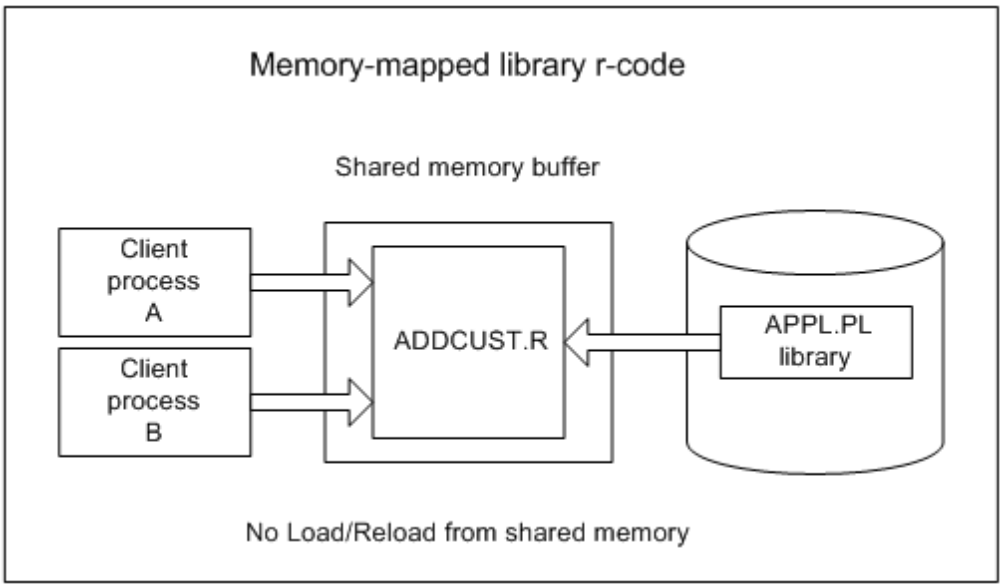
Note: The AVM swaps codepage-converted text segments for procedure libraries from the execution buffer to the r-code swap file and reloads the converted text segments from the swap file instead of the library when necessary.

Loading r-code from a memory-mapped library

After locating a member in a memory-mapped library, OpenEdge loads the member by mapping the library in shared memory where one or more clients can access it.

The following image shows the load operation when executing a member from a memory-mapped library.

Figure 7: Loading r-code from a memory-mapped library



As shown in the above image, when one or more client processes access a member in a memory-mapped library, they access the same segments in shared memory. Because OpenEdge executes the segments from the mapped library in shared memory (not local memory), no reloading is necessary. If you specify the PROLIB Swap (`-pls`) startup parameter with memory-mapped libraries, OpenEdge ignores it.

Note: The AVM swaps codepage-converted text segments for procedure libraries from the execution buffer to the r-code swap file and reloads the converted text segments from the swap file instead of the library when necessary.

Setting up a library

This section provides step-by-step instructions for setting up a library using the PROLIB utility.

To set up a library:

1. Create the standard library with the PROLIB utility. (All libraries must have a `.pl` extension.)
2. Add the r-code files to the library with the PROLIB utility.

3. If you are setting up a memory-mapped library, use the PROLIB utility to generate the memory-mapped library from the standard library.
4. Place the library in the PROPATH.

   For more information about the PROLIB utility, see Using the PROLIB utility on page 122. For more information about placing libraries in the PROPATH, see Libraries and PROPATH on page 120.

Running members from a library

The difference between running a procedure from an operating system file and running a member from a library is in how you reference it. You can reference both using either absolute or relative pathnames.

Using an absolute pathname

The most efficient but least flexible way to reference r-code in a library is by using an absolute pathname. When you reference a member using an absolute pathname, OpenEdge does not search the PROPATH. It looks for the member in the specified directory.

To run a procedure from an operating system file in a specific directory, reference the procedure using an absolute pathname. For example:

```
RUN /usr/appl/addcust.r
```

However, to run a member from a library in a specific directory, you must reference both the library and the member with an absolute pathname. For example:

```
RUN /usr/appl.pl<<addcust.r>>
```

When you reference a member in a specific library, you must enclose the member name in double angle brackets as shown.

Using a relative pathname

The most flexible but least efficient way to reference r-code in a library is by using a pathname relative to a location on the PROPATH. When you reference a member using a relative pathname, OpenEdge searches the PROPATH.

When you reference a member in an unspecified library using a relative pathname, OpenEdge searches each directory and library defined in the PROPATH until it finds the first occurrence of the procedure. For example:

```
RUN mydir/appl/addcust.r
```

When you reference a member in a specific library using a relative pathname, OpenEdge searches the PROPATH until it finds the first occurrence of the library. For example:

```
RUN mydir/appl.pl<<addcust.r>>
```

When you reference a member in a specific library, you must enclose the member name in double angle brackets as shown.

The most flexible way to find r-code is to specify relative pathnames for all procedures run in your application. This enables you to include a procedure in a library at deployment, but maintain separate procedure and r-code files during development. In a development environment, either place the library in the PROPATH after your development directories or omit it from the PROPATH entirely. In a production environment, place the library at or near the beginning of the PROPATH to avoid long searches.

Note: OpenEdge can only find r-code members in a procedure library. While it is technically possible to include other types of files in a procedure library, OpenEdge cannot find them while searching the PROPATH.

For more information about how libraries interact with PROPATH, see Libraries and PROPATH on page 120. For more information about the `RUN` statement, see *ABL Reference*.

Related functions

If a file is in a library, the `SEARCH` function returns the file's pathname using the following syntax:

Syntax

```
library-pathname<<member-name>>
```

For example:

```
/usr/dictionary/dictionary.pl<<dict.r>>
```

Other ABL functions make use of this syntax. The `LIBRARY` function parses the pathname and returns the name of the library (in this example, /usr/dictionary/dictionary.pl). If the argument to `LIBRARY` is not in the form *library-pathname<<member-name>>*, the `LIBRARY` function returns the `Unknown value (?)`. This is also true of the `MEMBER` function, which parses the pathname and returns the name of the member file (in this example, `dict.r`).

For more information about the `SEARCH`, `LIBRARY`, and `MEMBER` functions, see *ABL Reference*.

Libraries and PROPATH

The following rules govern how standard and memory-mapped libraries interact with PROPATH during an OpenEdge session:

- The first time you run a member from a standard or memory-mapped library that is specified in the PROPATH, OpenEdge locates the member by searching through each directory and library in the PROPATH until it finds the member. To search a library for a member, OpenEdge must open the library. When OpenEdge opens a standard library, it loads the procedure into local memory. When OpenEdge opens a memory-mapped library, it maps the library in shared memory.
- When searching through directories and libraries in the PROPATH, OpenEdge starts at the beginning of the PROPATH and searches each directory and library, in defined order, until it finds the member. Thus, placing the library at the beginning of the PROPATH improves performance.

- A library remains open until the end of your OpenEdge session or until you remove the library from the PROPATH. If you remove a library from the PROPATH while a member is still active (either running, or waiting for a subprocedure to return), OpenEdge displays an error message and the library remains open until you end your OpenEdge session. Otherwise, OpenEdge closes a standard library or unmaps a memory-mapped library.
- If you use a `SEARCH` or `RUN` statement to open a library that is not in the PROPATH, the library remains open until you end your OpenEdge session.
- If you use libraries, OpenEdge accesses r-code as if you had specified the Quick Request (`-q`) startup parameter. That is, OpenEdge searches the PROPATH only on the first reference to a member. Thereafter, if the member still resides in memory, in the local session compile file, or in an procedure library, OpenEdge reuses the member instead of searching the PROPATH again. For more information about the Quick Request (`-q`) startup parameter, see *Startup Command and Parameter Reference*.

Note: To ensure that all users access the same memory-mapped library in shared memory, each user must place the library in their PROPATH using the same library pathname.

For more information about running members from a library, see Running members from a library on page 119.

Memory and network considerations

Each library has an internal directory that OpenEdge uses to access members. OpenEdge loads this directory into memory when it opens the library. The directory stays in memory until the library is closed.

By default, OpenEdge tries to load the entire directory of each library into memory. In some cases, you might not have enough available memory to store a library's entire internal directory. Use the PROLIB Memory (`-plm`) startup parameter to limit the size of a standard library's internal directory in memory. This parameter allocates a 512-byte cache for a standard library's directory. This parameter is especially useful when running in limited memory with several libraries opened simultaneously. If you specify the PROLIB Memory (`-plm`) startup parameter with memory-mapped libraries, OpenEdge ignores it.

If you run OpenEdge over a network and place a standard library on a file server, remote reads from the library might take longer than reading a r-code file and storing it locally in the r-code swap file. Use the PROLIB Swap (`-pls`) startup parameter to store standard library members locally in the r-code swap file. If you specify the PROLIB Swap (`-pls`) startup parameter with memory-mapped libraries, OpenEdge ignores it.

For more information about the PROLIB Memory (`-plm`) and PROLIB Swap (`-pls`) startup parameters, see *Startup Command and Parameter Reference*.

Using the PROLIB utility

The PROLIB utility allows you to create and maintain standard libraries and generate memory-mapped libraries. This is the syntax to start the PROLIB utility:

Operating system	Syntax
UNIX Windows	`prolib library-name parameter [ file-name . . . ]` `[ parameter [ file-name . . . ] . . . ]`

library-name

Specifies the name of a procedure library. The library name must have a `.pl` extension.

parameter

Specifies what action to take on the library. The following table lists the parameters and their descriptions.

Table 26: PROLIB utility parameters

Parameters	Description
`-create`	Creates a new standard library.
`-makeshared`	Generates a memory-mapped library from a standard library.
`-add`	Adds files to a standard library.
`-replace`	Replaces files in a standard library.
`-delete`	Deletes files from a standard library.
`-list`	Lists library information such as library name, format, code page, and file contents.
`-extract`	Extracts files from a standard library. This parameter creates a copy of a file from the library outside of the library.
`-yank`	Extracts files from a standard library and places them in the current working directory.
`-compress`	Compresses a standard library by making a copy of it.
`-nowarn`	Suppresses any warning message that might occur during the operation of the primary parameters. If you add a file to a standard library with the -add and -nowarn parameters, and the file already exists in the library, PROLIB replaces the file.
`-pf`	Allows you to supply command-line arguments in a parameter file. You cannot use -pf in a parameter file.

Parameters	Description
-verbose	Directs PROLIB to display processing information that is ordinarily suppressed.
-date	Specifies the format of the date as it appears when you use the -list parameter.

When specifying a parameter, you do not have to type the complete parameter name. You can type the minimally unique string for each parameter (for example, -l for -list and -e for -extract).

The -nowarn and -verbose parameters modify the behavior of the -create, -makeshared, -add, -replace, -delete, -list, -extract, and -yank parameters.

You can place the -nowarn, -pf, and -verbose parameters anywhere on the command line. They affect the processing of all other specified parameters.

You must place the -create parameter before all other parameters. PROLIB processes parameters in left-to-right order as they appear on the command line. If an error occurs during the PROLIB command, PROLIB terminates. This behavior occurs so that options specified later in the command line, which might depend on the failed option, do not execute.

You cannot use the -add, -replace, -delete, -extract, -yank, or -compress parameters with a memory-mapped library (that is, when you specify a memory-mapped library in *library-name*).

file-name

Specifies the name of an r-code file, or a memory-mapped library file when using the -makeshared parameter.

Using wild cards

The -add, -replace, -extract, -list, -delete, and -yank parameters accept wild card arguments. Wild card arguments can contain only the ? or * wild card characters. Depending on the parameter you want to use, you must specify these arguments in one of two ways:

- Using your operating system's regular wild card conventions
- Escaping the wild cards

The -add and -replace parameters act on operating system files. PROLIB uses system calls to copy the files. Thus, when you use these parameters, use your operating system's standard conventions to specify wild cards.

The -extract, -list, -delete, and -yank parameters act on members that are already in a library. PROLIB does not use system calls to act on the files, but instead uses its own internal code. On UNIX, you must escape your wild card arguments by either enclosing them in quotes (for example, `prolib app.pl -delete "sys*.r"`) or escaping the wild card characters individually (for example, `prolib app.pl -delete sys\*.r`). Your operating system might use different techniques to escape wild cards. In Windows, you do not need to escape wild card arguments because the operating system does not expand them before passing them to PROLIB.

Creating a standard library

This is the syntax for the PROLIB command to create a standard library:

Operating system	Syntax
UNIX Windows	`prolib library-name` `  -create [ -codepage codepage-name ]`

`library-name`

Specifies the name of the library you want to create.

All libraries must have a `.pl` extension. You can add a library to the PROPATH environment variable by specifying the file's absolute pathname or its pathname relative to the current working directory.

`codepage-name`

Specifies the name of the code page for the library you want to create.

A library can contain members with different code pages. All members retain their respective code pages.

If you do not specify the `-codepage` parameter at creation time, OpenEdge marks the procedure library with the code page specified in the Internal Code Page (-cpinternal) startup parameter.

For information about code page compatibility, see Code page compatibility on page 131.

Generating a memory-mapped library

This is the syntax for the PROLIB command to generate a memory-mapped library from a standard library:

Operating system	Syntax
UNIX Windows	`prolib standard-library-name` `   -makeshared mapped-library-name`

`standard-library-name`

Specifies the name of the standard library from which to generate the memory-mapped library. The library name must have a `.pl` extension. You can add a library to the PROPATH environment variable by specifying the file's absolute pathname or its pathname relative to the current working directory.

Save the standard library; it is the source of modification for the memory-mapped library. Each time you modify the standard library, you must regenerate the memory-mapped library.

`mapped-library-name`

Specifies the name of the memory-mapped library you are generating. The library name must have a `.pl` extension.

Note: You cannot modify a memory-mapped library directly. To modify a memory-mapped library, you must modify its source standard library and regenerate the memory-mapped library.

PROLIB generates a memory-mapped library with the standard library's code page. All text segments in all members retain their respective code pages.

For information about code page compatibility, see Code page compatibility on page 131.

Adding files to a standard library

This is the syntax for the PROLIB command to add files to a standard library:

Operating system	Syntax
UNIX Windows	`prolib` *`library-name`* `-add` *`file-name`* **[** *`file-name`* **]** **. . .**

`library-name`

Specifies the name of the library where you want to add the file.

`file-name`

Specifies the pathname of the file. You can add a file to a library by specifying the file's absolute or relative pathname.You can add a file to the PROPATH environment variable by specifying the file's absolute pathname or its pathname relative to the current working directory.

When you add a file to a library, the library entry for that file retains the entire pathname. For example:

```
prolib newlib.pl -add /usr/apps/proc1.r
```

The pathname `/usr/apps/proc1.r` appears in the library's table of contents. You can display the library's table of contents with the `-list` parameter.

When you extract a file from a library, PROLIB copies the file into the directory specified by the pathname (in this example, `/usr/apps`). See Extracting files from a standard library on page 128 for more information.

If you try to add a file that already exists, PROLIB displays an error message. However, if you specify the `-nowarn` parameter, PROLIB replaces the existing file. See Replacing files in a standard library on page 126 for more information.

Replacing files in a standard library

This is the syntax for the PROLIB command to replace one or more files in a standard library:

Operating system	Syntax
UNIX Windows	`prolib library-name -replace file-name [ file-name ] . . .`

library-name

Specifies the name of the library with the file you want to replace.

file-name

Specifies the name of the file.

When you replace a file, the system overwrites the old file with a file of the same name. Therefore, the pathname you supply for the file has to be the same as the pathname you used originally to add the file to the library. If you try to replace a file that does not exist, PROLIB displays an error message. However, if you specify the `-nowarn` parameter, PROLIB adds the existing file to the library. See Adding files to a standard library on page 125 for more information.

Deleting files from a standard library

This is the syntax for the PROLIB command to delete files from a standard library:

Operating system	Syntax
UNIX Windows	`prolib library-name -delete file-name [ file-name ] . . .`

library-name

Specifies the name of the library with the file you want to delete.

file-name

Specifies the name of the file. You can specify more than one file at a time.

Listing the contents of a library

This is the syntax for the PROLIB command to list the contents of a library:

Operating system	Syntax
UNIX Windows	`prolib library-name -list [file-name] . . .`

library-name

Specifies the name of the library with the contents you want to list.

file-name

Specifies that PROLIB should list information about that file only; *file-name* has to be the same absolute or relative pathname as when you originally added the file. You can specify more than one file at a time.

The following table lists the information that appears in the header section of the output.

Table 27: List (-list) parameter library information

Information	Description
Library Name	The name of the library.
Code Page	The code page in which the library was created.
Format	The format of the library. The valid library formats are STANDARD and MAPPED.

The following table lists the information that appears in the output for each file in the library.

Table 28: List (-list) parameter file information

Information	Description
Name	The name of the file.
Size	The size of the file in the library, in bytes.
Type	The file type. PROLIB recognizes two file types: R (r-code file type) and O (any other file type). Although libraries are specifically designed to hold r-code files, you can place any type of file in them.
Offset	The distance, in bytes, of the start of the file from the beginning of the library.

Information	Description
Modified	The date and time the file was last modified.
Added To Lib	The date and time the file was entered into the library. You can also change the format of the dates as they appear in the **Modified** and **Added To Lib** fields with the -date parameter, where *date-format* is three letters: m (month), d (day), and y (year). To specify a format, place the letters in the order you want the months, days, and years to appear on screen. The default is mdy.

Extracting files from a standard library

This is the syntax for the PROLIB command to extract files from a standard library:

Operating system	Syntax
UNIX Windows	`prolib` *library-name* `-extract` *file-name* **[** *file-name* **] . . .**

library-name

Specifies the name of the library with the file you want to extract.

file-name

Specifies the name of the file. You can specify more than one file at a time.

Note: You **must** create all subdirectories required by a library **before** trying to extract files from the library. You can see what directories and subdirectories a library needs by using the PROLIB `-list` parameter to view the contents of the library.

When you extract a file from a library, PROLIB copies the file and places it in its original directory. Depending on how you add a file to a library, PROLIB extracts it differently. For example:

- If you added the file by specifying a relative pathname, PROLIB searches your current working directory for the subdirectory indicated in the pathname. For example, if you added a file by using the relative pathname `apps/proc1.r`, PROLIB searches your current working directory for the subdirectory `apps`. If the directory does not exist, PROLIB displays an error message.
- If you added the file by specifying its absolute pathname, PROLIB searches for the original directory. If the directory does not exist, PROLIB displays an error message.

Note: If you specify the `-yank` parameter instead of `-extract`, OpenEdge strips everything but the filename from the pathname and places the file in your current directory.

If the file you are extracting (or yanking) already exists in the directory outside of the library, PROLIB displays the following message (where *file-name* is the pathname of the file you are extracting or yanking):

```
File file-name already exists. Do you want to replace it?
Please answer 'yes' or 'no' or 'rename' to assign new name:
```

If you answer yes, PROLIB overwrites the file. If you answer no, PROLIB does not overwrite the file. If you answer rename, PROLIB displays the following message (where *file-name* is the pathname of the file you are extracting or yanking):

```
Enter new name to use for extracted file file-name -->
```

After you enter the pathname, PROLIB gives the file the new pathname and places it in the appropriate directory.

Extracting files to your current directory

When extracting a file with the `-yank` parameter, PROLIB discards everything but the filename from the file's pathname. For example, if you added a file using the pathname `/usr/apps/proc1.r`, PROLIB discards `/usr/apps` from the pathname and places `proc1.r` in your current working directory.

This is the syntax for the PROLIB command to extract files from a library using the `-yank` parameter:

Operating system	Syntax
UNIX Windows	`prolib` *library-name* `-yank` *file-name* **[** *file-name* **] . . .**

library-name

Specifies the name of the library with the file you want to extract.

file-name

Specifies the name of the file. You can specify more than one file at a time.

Compressing a standard library

This is the syntax for the PROLIB command to compress a standard library:

Operating system	Syntax
UNIX Windows	`prolib` *`library-name`* `-compress`

`library-name`

Specifies the name of the library you are compressing. The `-compress` parameter removes extra spaces that occur in the library as a result of repeated adds or deletes.

To compress a library, PROLIB creates a temporary file that requires an area of disk space equal to the size of the compressed library. Before compressing a library, make sure you have enough disk space for the temporary file. Temporary files have names that begin with the letters `PLB`, followed by numbers and possibly letters.

If your system goes down during a compress operation, the temporary file might remain on your disk. Delete this file to free up disk space. The original library will not be damaged.

Repeated adds create empty spaces in the library over time. To minimize the rate at which these empty spaces are created, you can use one command instead of many. For example, instead of entering the following commands:

```
prolib test.pl -add test.r
prolib test.pl -add test2.r
prolib test.pl -add test3.r
```

Enter this command:

```
prolib test.pl -add test*.r
```

You can use the `-list` parameter to determine how much space in your library is unused.

PROLIB command examples

This section provides several PROLIB command examples.

The following example creates a standard library called `app.pl` in your current directory:

```
prolib app.pl -create
```

The following example adds all the r-code files in the current directory that begin with the letters "sys" to the standard library `app.pl`. In this example, the r-code files are in a directory, so the operating system processes the wild card argument, as shown:

```
prolib app.pl -add sys*.r
```

The following example deletes all members ending with the letters "dev.r" from the standard library `app.pl` and lists all of the members in the library. In this example, the members are in the `app.pl` library, so PROLIB processes the wild card argument. This requires escaping the `*dev.r` argument. Your operating system might use different techniques to escape wild cards. On UNIX, this means placing quotes around it, as shown:

```
prolib app.pl -delete "*dev.r" -list
```

The following example replaces the members `appmenu.r` and `apprept.r` in the standard library app.pl, then compresses the library to free up unused disk space:

```
prolib app.pl -replace appmenu.r apprept.r -compress
```

The following example creates a standard library called `app.pl` in your current directory, adds all the r-code files in the current directory that begin with the letters "sys" to the library `app.pl`, and generates a memory-mapped library called `app_map.pl`:

```
prolib app.pl -create -add sys*.r -makeshared app_map.pl
```

Code page compatibility

The Internal Code Page (`-cpinternal`) startup parameter determines the internal code page that OpenEdge uses in memory. The code page in which r-code is compiled and the internal code page of the session in which the r-code runs must be compatible. That is, the r-code code page must be either the internal code page, undefined, or convertible to the internal code page.

If the code page of the r-code, from either an operating system file, a memory-mapped procedure library, or a standard procedure library, is not the same as the session's internal code page, OpenEdge performs an in-memory conversion of the r-code text segments to the session's internal code page.

Note: If you use the R-code In Code Page (`-cprcodein`) startup parameter to specify a code page for reading r-code text segments, OpenEdge reads the text segments as if they were written in that code page, even if the text segments were written in a different code page.

For more information about the Internal Code Page (`-cpinternal`) and R-code In Code Page (`-cprcodein`) startup parameters, see *Startup Command and Parameter Reference*.

7

Managing Print Devices

System administrators are responsible for setting up and maintaining the output devices required by an application. Most OpenEdge® applications access printers to print reports or other output. This chapter provides an overview of how to set up print devices on OpenEdge-supported operating systems. It also describes how to use OpenEdge to set up and control output routing and printing characteristics.

For details, see the following topics:

- Printing in OpenEdge
- Printing in Windows
- Setting up output-routing
- Printing on UNIX
- Configuring a printer

Printing in OpenEdge

Most operating systems support two general printing techniques: direct printing and spooled printing. *Direct printing* sends application output directly to a printer set up on the operating system. This technique of printing is frequently found on single-user systems. However, you can use direct printing in other situations (for example, printing tickets and special forms).

Spooled printing sends application output to a temporary file and places a print request in a print queue set up on the operating system. Spooling permits you to queue a print request and continue processing; the system executes the print request when a printer becomes available. A spooled device can help balance the demand on line printers if you are running applications on a time-shared system.

OpenEdge runs with most printers and, depending on the operating system in use, makes default assumptions about printer setup. OpenEdge uses the printer port or spooler named at execution time, not at compile time. Thus, precompiled OpenEdge applications work regardless of the printer port or spooler in effect.

This section provides an overview of the ABL (Advanced Business Language) language elements you can use to direct application output, which includes the:

- `OUTPUT TO` statement
- `OUTPUT THROUGH` statement

OUTPUT TO statement

By default, all OpenEdge output is directed to the terminal. The `OUTPUT TO` statement allows an application to redirect output to other output devices. The following is a partial syntax example of the `OUTPUT TO` statement:

Syntax

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] TO
  {     PRINTER [ printer-name ] |  opsys-file
     |  opsys-device
     |  TERMINAL
     |  VALUE ( expression )
     |  "CLIPBOARD"
  }
        .
        .
        .
  [ PAGED ] [ PAGE-SIZE { constant | VALUE ( expression ) } ]
        .
        .
        .
```

Use the `OUTPUT TO` statement with the `PRINTER` option to send output to the default print device on the current operating system. For example:

```
OUTPUT TO PRINTER.
```

On UNIX, the `OUTPUT TO PRINTER` statement sends output to the default spooler. To use a spooler other than the default, start OpenEdge using the Printer (`-o`) startup parameter.

In Windows, the `OUTPUT TO PRINTER` statement sends output to the printer defined in default print context. The default print context is the set of values that defines the default printer and setup for that printer in Windows. If there is no default print context, OpenEdge uses the printer control settings from the current environment.

Use the `SYSTEM-DIALOG PRINTER-SETUP` statement to display the Windows Print dialog box and let the user change the default print context at runtime. Use the `PRINTER-NAME` attribute of the `SESSION` system handle to change the printer name in the default print context without user intervention. You can change the system default printer from the Windows Control Panel.

Use the OUTPUT TO PRINTER statement with its various options to override the default print context for a specific print job. For example, send output to a printer other than the default printer by using the following syntax:

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] TO PRINTER printer-name
```

OpenEdge assumes that all printers can handle hard-coded form feed characters (**CTRL+L** or 0C Hex). The OUTPUT TO PRINTER statement automatically paginates OpenEdge output on the default printer. To paginate output on a printer, other than the default printer, use the OUTPUT TO PRINTER statement with the PAGED or PAGE-SIZE option.

Use the Printer (-o) startup parameter to specify a default printer to use when processing the OUTPUT TO PRINTER statement in procedures. The specified printer must be set up on the current operating system.

For a complete description of the OUTPUT TO PRINTER statement, the SYSTEM-DIALOG PRINTER-SETUP statement, and the SESSION system handle, see *ABL Reference*. For more information about the Printer (-o) startup parameter, see *Startup Command and Parameter Reference*. For more information about streams, see *OpenEdge External Programming Interface Reference*.

OUTPUT THROUGH statement

The OUTPUT THROUGH statement allows an application to redirect output to a process that OpenEdge starts. On UNIX, this statement allows you to pipe OpenEdge output to a UNIX process. The following is a partial syntax example of the OUTPUT THROUGH statement:

Syntax

```
OUTPUT [ STREAM stream | STREAM-HANDLE handle ] THROUGH
  { program-name | VALUE ( expression ) } [ argument | VALUE ( expression ) ] . . .
     .
     .
     .
```

You can also use the OUTPUT THROUGH statement to access spooling capabilities on UNIX systems. For example, this command sends output to the printer named lw on a UNIX system:

```
OUTPUT THROUGH lpr -Plw.
```

For a complete description of the OUTPUT THROUGH statement, see *ABL Reference*. For more information about streams, see *OpenEdge External Programming Interface Reference*.

Printing in Windows

In Windows, you can choose either of the following two printing modes:

- Default mode
- Windows Passthrough Printing (WPP)

In default mode, OpenEdge uses the Windows device drivers. This mode is sufficient for all basic printing that does not use any printer access control codes, such as those sent by `PUT` or `DISPLAY` statements.

In WPP mode, OpenEdge bypasses the Windows device drivers. This means you can send control codes (such as POSTSCRIPT commands) directly to the printer with the `PUT` or `DISPLAY` statements. To enable windows passthrough printing for a session, use the Windows Passthrough Printing (`-Wa -wpp`) startup parameter. You can only use this parameter on the command line.

You cannot use WPP mode with a POSTSCRIPT printer to print OpenEdge standard reports, such as those produced by the Data Dictionary.

Note: When using windows passthrough printing in Windows95, you must change the Spool Data Format from bmp spooling to RAW spooling. Use the Control Panel to access the Printer Properties for the printer, then select Spool Setting from the Details tab of the Printer Properties tab. Change the Spool Data Format to RAW spooling.

For a complete description of the `PUT` and `DISPLAY` statements, see *ABL Reference*. For more information about the Windows Passthrough Printing (`-Wa -wpp`) startup parameter, see *Startup Command and Parameter Reference*.

Setting up output-routing

For systems with multiple printers, you might want to set up an output-routing scheme. Output-routing schemes direct application output to different devices based on criteria that you specify. Since OpenEdge uses the printer port or spooler named at execution time and not at compile time, you can set up output-routing schemes to send application output to different printers based on user identification, the type of procedure the user runs, or some other criteria.

When setting up an output-routing scheme, examine the application and system requirements. For example, your system users might want to direct output to different printers. In this case, you would have to set up an output-routing scheme based on the user identification. An application might have reports that require different types of printers. For example, one report might require a 132-character printer and another report might require an 80-character printer. In this case, you would have to set up an output-routing scheme based on the report type. Another output-routing scheme could direct output to different printers based on both the user and the type of report.

You can set up an output-routing scheme in the following ways:

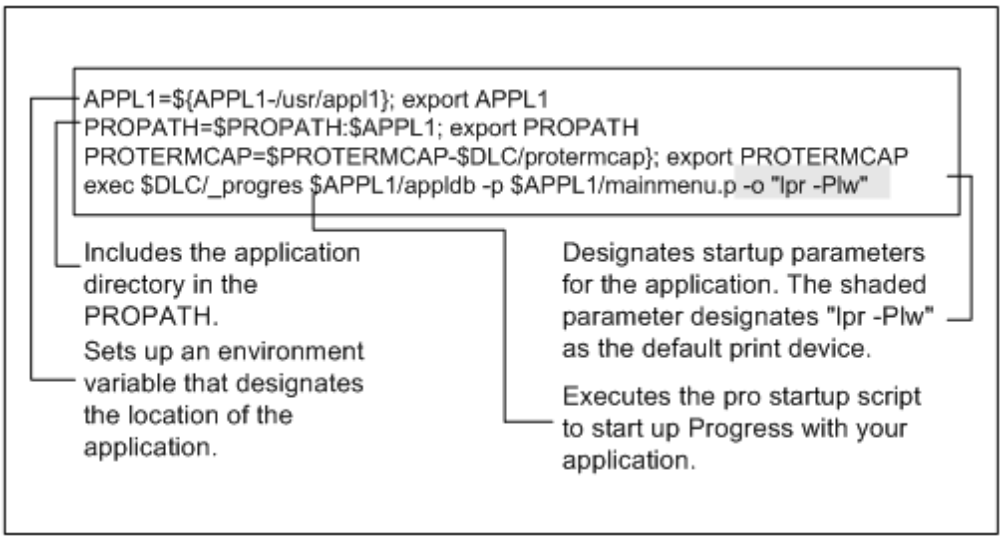
- Using a startup procedure on page 136
- Using an output-routing table on page 137
- Using interactive output-routing on page 138

Using a startup procedure

The simplest output-routing scheme consists of setting up an OpenEdge application startup procedure for each printer on a system. Each startup procedure could contain the Printer (-o) startup parameter to designate a particular printer as the default printer for the current OpenEdge session.

The following figure shows a sample application startup procedure on UNIX. This application startup procedure designates lpr -Plw as the default print device for the current OpenEdge session and starts up an OpenEdge application located in the `/usr/appl1` directory that access a database called `appldb`.

Figure 8: Sample application startup procedure (UNIX)



The above figure shows a sample application startup procedure on Windows. This application startup procedure designates LPT2 as the default print device for the current OpenEdge session and starts up an OpenEdge application located in the `/usr/appl1` directory that access a database called `appldb`.

Sample application startup procedure (Windows)

```
echo off
set APPL1=\usr\APPL1
set PROPATH=%PROPATH%;%APPL1%
%DLC%\_progres -D 20  -o LPT2 -p
\usr\APPL1\mainmenu.p appldb
```

After the startup procedures are set up for each system printer, you can designate users to use different startup procedures depending on which output destination they want. For easy user access and maintenance, all of the startup procedures should be located in the directory where the system software is located.

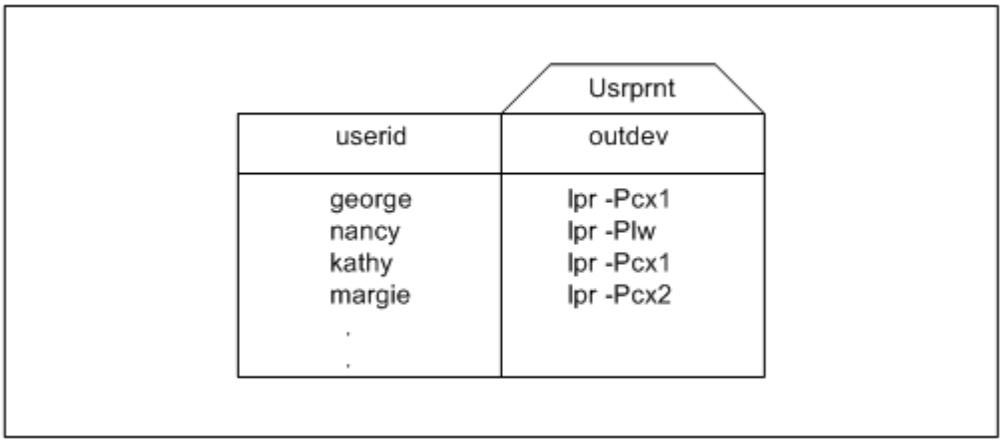
Using an output-routing table

More complex output-routing schemes, such as routing output based on the user ID, require the creation of several related ABL tables and an include file that can access these tables and select the proper output command for the particular user.

Just as you can use ABL tables (`_User`) to set up security for OpenEdge applications, you can also use ABL tables for output routing. If you want to route application output to different printers based on user ID, set up a table in the application database containing each of the valid user IDs and the corresponding print commands.

T shows a sample output-routing table.

Figure 9: Sample output-routing table



The database table shown in the figure contains records that have a userid field and an outdev field containing an operating system output device. The userid field is a unique primary index for the table. You have to keep the Usrprnt table up to date with a record for each new user on your system.

After setting up the database table, create an include file that determines the current user ID and then searches the Usrprnt table for a matching user ID and corresponding output device. For example:

```
FIND Usrprnt WHERE Usrprnt.userid = USERID NO-ERROR.
  IF AVAILABLE Usrprnt THEN
    OUTPUT THROUGH VALUE(Usrprnt.outdev).
  ELSE
    OUTPUT TO PRINTER.
```

The include file searches the Usrprnt table for a record that has a user ID that matches the user ID established for the current application session. If there is no user ID match or established user ID for the application session, OpenEdge uses the default system printer. If there is a match, OpenEdge sends the application output to the output device contained in the corresponding outdev field.

The `VALUE` option of the `OUTPUT THROUGH` statement allows you to designate a substitutable output device. The `USERID` function allows you to capture system user IDs on UNIX. For more information about the `_User` table, see *Manage the OpenEdge Database*.

After creating an include file to route application output, replace all `OUTPUT` statements in your application with the include file. For example:

```
{output1.i}.
FOR EACH Customer NO-LOCK:
  PUT Customer.CustNum Customer.Name Customer.Address Customer.City     Customer.State
 Customer.PostalCode.
END.
```

Using interactive output-routing

You can also set up interactive output-routing schemes that let application users select an output device at run time. In Windows, for example, you can change the destination of the `OUTPUTTO PRINTER` statement at run time by using the `SYSTEM-DIALOGPRINTER-SETUP` statement.

Printing on UNIX

Refer to your UNIX system documentation for information about printing on UNIX.

Configuring a printer

Many printers have a set of control sequences that you can use to specify different print characteristics, such as compressed printing or italics typeface. These control sequences often involve special characters that can be represented by their octal (base 8) equivalent. To denote these octal codes, precede the three octal digits with an escape character. On UNIX, the escape character is a tilde (~) or a backslash (\). The following table lists the standard control sequences using a tilde as the escape character.

Table 29: Standard printer control sequences

Sequence	Interpreted as	Comments
`~nnn`	Single character	Use to send a single character to the printer, where nnn is an octal code between 000 and 377. You must specify all three digits.
`~t`	Tab character	Use to send the 011 octal code to the printer.
`~r`	Carriage return	Use to send the 015 octal code to the printer.
`~n`	Line feed	Use to send the 012 octal code to the printer.
`~E`	Escape	Use to send the 033 octal code to the printer.
`~b`	Backspace	Use to send the 010 octal code to the printer.
`~f`	Form feed	Use to send the 014 octal code to the printer.
`~"`	"	Use within quoted strings as an alternative to two quotes ("").
`~\`	\	Use to send the backslash character to the printer.
`~~`	~	Use to send the tilde character to the printer.

The `PUT` statement with the `CONTROL` option allows you to specify a control sequence that you want to send to a printer. The `PUT` statement has the following syntax:

Syntax

```
PUT [ STREAM stream-name]
  CONTROL "ctrl-sequence" "ctrl-sequence" . . .
```

In the following example, the control sequence ~017 turns on the compressed printing feature of the current print device:

```
PUT STREAM a CONTROL "~017".
```

The control sequences sent to the output destination do not affect the current line, page counters, and positions maintained within OpenEdge. For more information about the `PUT CONTROL` statement, see *ABL Reference*.

Creating a printer-control table

It is important to note that control sequences are hardware dependent. Applications with hard-coded printer-control sequences are not easily transportable. You cannot change the print devices that the application accesses without changing the hard-coded printer-control sequences in the application. To bypass this obstacle, create a printer-control table which contains a record for each printer on your system. Each record in the printer-control table must contain a field for the name of the print device and several other fields containing a corresponding set of standard control sequences.

There is no easy way to employ an ABL `UPDATE` statement to interpret the printer-control sequences and enter them into a printer-control table. For example, the ABL `UPDATE` statement interprets the octal sequence ~017 as the character string, as shown:

```
[TILDE][ZERO][ONE][SEVEN]
```

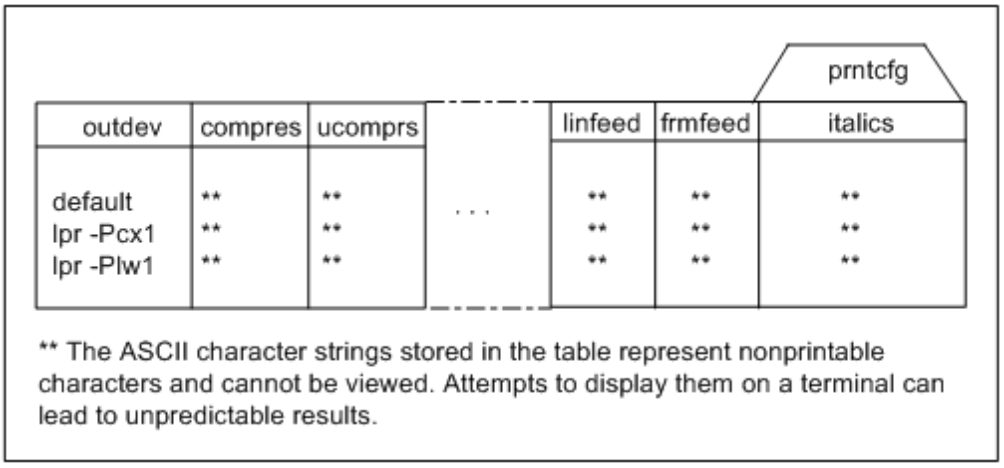
Instead of the proper code sequence:

```
[CONTROL-O]
```

You can create a system administration procedure that lets you convert printer-control sequences into the appropriate ASCII character strings, and store the printer-control sequences as ASCII character strings in the fields of a printer-control table.

Keep the printer-control table up to date with a record for each printer on your system. The following image shows a sample printer-control table.

Figure 10: Sample printer-control table



The field containing the print device name must index the table. In this sample control table, the outdev field indexes the table.

To access the proper control sequences for a printer from a printer-control table, set up an include file to determine the print device used in the application, then substitute the corresponding control sequences into the PUT CONTROL statements of the application. For example:

```
FIND Usrprnt WHERE Usrprnt.userid = USERID NO-ERROR.
  IF AVAILABLE Usrprnt
  THEN DO:
    OUTPUT THROUGH VALUE(Usrprnt.outdev).
    FIND Prntcfg WHERE Prntcfg.outdev = Usrprnt.outdev NO-ERROR.
  END.
  ELSE DO:
    OUTPUT TO PRINTER.
    FIND Prntcfg WHERE Prntcfg.outdev = "default".
  END.
```

The include file searches the Usrprnt table to determine the output device (Usrprnt.outdev) for the current user ID. After determining the output device, the include file searches the printer-control table to find the appropriate control sequences for the current printer. Notice that the printer-control table contains a default entry for the default printer.

The PUT CONTROL statements in your application should use a field in the printer-control table or a variable name to access the proper control sequence for the current printer. This allows you to change print devices without recoding the printer-control sequences for all the PUT CONTROL statements in your application. For example:

```
{output2.i}.
PUT CONTROL Prntcfg.compres.
FOR EACH Customer NO-LOCK:
  PUT Customer.CustNum Customer.Name Customer.Address Customer.City     Customer.State
 Customer.PostalCode.
END.
PUT CONTROL Prntcfg.ucomprs.
```

In this procedure, the first PUT CONTROL statement turns on compressed printing, and the second PUT CONTROL statement turns off compressed printing for the current print device.

II

Deployment Considerations

For details, see the following topics:

- Choosing a Code Format
- Initial Deployment
- Upgrades
- Deployment Topics and Tasks

8

Choosing a Code Format

Choosing a code format is the fundamental decision that you must make about deploying an application. This chapter describes the three types of code format, the OpenEdge® product requirements for those code formats, and the advantages and disadvantages of each format.

For details, see the following topics:

- Unencrypted source
- Encrypted source
- R-code

Unencrypted source

Unencrypted source is uncompiled source that users can read and edit with any standard text editor. Users with full compilation capabilities can read and modify unencrypted source. Thus, when you choose to deploy using unencrypted source, you give users unrestricted access to your application code and application database.

The following sections provide information about the requirements for using unencrypted source, and discuss some advantages and possible disadvantages to deploying unencrypted source.

Development product requirements for unencrypted source

Developers must be able to repeatedly modify, compile, and test your application to ensure that your application behaves as expected. You must therefore have a full development OpenEdge product on your development platforms. The 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, and WebSpeed Workshop products are all full development products.

User product requirements for unencrypted source

To read, modify, compile, and execute unencrypted source, the user must have a full development OpenEdge product (that is, the 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, or WebSpeed Workshop). The Query/Results Client product is not a full development product because it can compile only read-only application files (source files that contain no code that modifies the database). Also, products that contain client execution engines—for example, the Database Server or Client Networking products—cannot compile unencrypted source.

Advantages of unencrypted source

Unencrypted source offers several advantages over other code formats:

- **Application extendability** — Since users have full access to your source code and full compilation capabilities, they can modify your application code any way they want.
- **Less to manage at development site** — Other code formats require you to manage multiple code trees and might require you to keep multiple copies of your application database.
- **Simpler initial deployment** — When you choose unencrypted source, deploying your application is usually easier than with other code formats. Initial Deployment on page 151 describes the steps required for each code format.
- **Simpler upgrades** — When you choose unencrypted source, upgrading your application is usually less complex than with other code formats. Upgrades on page 161 describes the steps required for each code format.
- **Easier maintenance** — Since users have full development environments, it is relatively easy to modify or enhance your application at their sites.
- **Full portability** — Both unencrypted and encrypted source are completely portable across platforms and user interfaces (character mode and GUI), provided you have followed the necessary guidelines for writing portable applications. Note that r-code is not portable across user interfaces.
- **Less space used on storage media** — Unencrypted source files usually occupy much less storage space than r-code files, decreasing their physical media shipment costs. (Encrypted source files also have this advantage over r-code files.)

Possible drawbacks of unencrypted source

Unencrypted source has several possible drawbacks, including:

- **Does not prevent code modifications** — Users can modify the application source, which can potentially cause difficulty when applying application upgrades.
- **Does not provide application source code protection** — Unencrypted source does not protect your application from the user. Many Independent Software Vendors (ISVs) do not want the user to be able to see their code.
- **Does not prevent database schema modifications** — Many ISVs do not want users to modify their application database. If you deploy using unencrypted source, you lose the ability to prevent unwanted changes. Users can modify your application and application database any way they want.
- **More difficult internationalization** — Only r-code can contain multiple text segments for different spoken languages.
- **More difficult to support** — Users might create bugs when they modify the source code.

Encrypted source

Encrypted source is source code that is encrypted with the XCODE utility (provided with OpenEdge). The user cannot make changes to encrypted source code because it is not readable by the user, so encrypting the source code protects your application.

The following sections provide information about the requirements for using encrypted source and discuss some advantages and possible disadvantages to deploying encrypted source.

More documentation for XCODE can be found in the XCODE utility on page 7.

Development product requirements for encrypted source

The requirements for encrypted source are the same as for unencrypted source. However, you also need one copy of OpenEdge (which contains the XCODE utility) for the development platform. You do not need a copy of XCODE for the different target platforms.

User product requirements for encrypted source

Any OpenEdge client, server, or standalone database product can compile and run encrypted source.

Advantages of encrypted source

Encrypted source code has several possible advantages over other code formats, including:

- **Application protection** — Because encrypted source cannot be read by users, it protects your application from users by preventing them from making changes. Encrypted source shares this advantage with r-code, which also is not user-readable.
- **Less to manage at development site** — Encrypting source code requires storing two code trees: the unencrypted source tree and the encrypted source tree (which must be kept in a parallel directory structure). While this is twice what is required with unencrypted source, it is still less than what is required for r-code.
- **Full portability** — Like unencrypted source, encrypted source is completely portable across platforms and user interfaces (character mode and GUI). However, r-code is not portable across user interfaces.
- **Compile-time flexibility** — Because encrypted code is not precompiled, your application can take advantage of compile-time functionality. For example, you might want to pass arguments to include files to determine sort order, field lists, etc. You can take advantage of this behavior at the user site, allowing users to specify their preferences before compilation. This option is not available with r-code, which you must compile at the development site.
- **Flexible index definitions** — If users have full development OpenEdge products, they can modify their indexes without invalidating your application code (provided that you have not explicitly named indexes in your code). This flexibility is not available with r-code, since r-code that specifically references a new or changed index does not work. Because the index Cyclic Redundancy Check (CRC) value is checked at run time, not all r-code will be affected.
- **Less space used on storage media** — Encrypted source files usually occupy much less storage space than r-code files, making your physical media shipment smaller and less expensive. (Unencrypted source files also have this advantage over r-code files.)
- **Encryption keys** — Encrypted source allows you to specify different encryption keys for different code trees or subsystems within a code tree. Therefore, it is possible to deploy an entire application at once, but only give the user encryption keys for certain parts of the application. This way, you can upgrade the user's application without physically distributing code a second time.

Possible drawbacks of encrypted source

Encrypted source code has the following possible drawbacks:

- **Does not prevent index modifications** — When compared to r-code, encrypted source is less restrictive because it allows modifications to the database's indexes. Whether this is an advantage or disadvantage depends on whether you want to give the user the flexibility to change indexes. If your code explicitly specifies indexes in its search criteria, user modifications to your indexes might invalidate your code.
- **The Encrypted Compiler Mode (-rx) parameter changes the capabilities of the Compiler** — When you start OpenEdge with the `-rx` parameter, ABL (Advanced Business Language) can compile only encrypted source, not unencrypted source. If users have the 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, or WebSpeed Workshop, they must start OpenEdge again without the `-rx` parameter to write and compile their own procedures.

R-code

R-code, or run-time code, is precompiled code. When you compile a source procedure, ABL places the compiled version of the procedure into a file that has the same name as the source procedure with a `.r` extension. For a complete discussion of r-code, see R-code Features and Functions on page 189.

Like encrypted source, r-code is not readable to the human eye and therefore protects your application from users attempting to make changes.

Developer product requirements and r-code portability

R-code is portable between any two environments if the following are true:

- The user interface is the same—MS-Windows or character mode.

Note: If a procedure contains no code that writes to the screen, its r-code is portable across user interfaces. For example, this is typical of database schema trigger procedures.

- The database type (the DataServer or combination of DataServers connected during compilation) is the same.
- The r-code version is the same.

For more details about these requirements and how they affect r-code portability, see R-code Features and Functions on page 189.

User product requirements

Any OpenEdge client, server, or standalone database product can run r-code.

Advantages of r-code

R-code provides the following advantages over other code formats:

- **Speed** — This format is faster because the r-code does not have to compile at run time.
- **Application protection** — Like encrypted source, r-code protects your application from possible user changes.

- **Good for internationalization** — R-code is the only code format that can contain multiple text segments. This means that the same application can contain strings for multiple spoken languages (French and German, for example). For more information about translating OpenEdge applications, see *OpenEdge Development: Translation Manager*.
- **Prevents database modifications** — CRC-based r-code is more tightly coupled to the database than encrypted or unencrypted source. For each table in the database, ABL calculates a CRC value that represents the structure of the table. A CRC value ensures that the tables in the database match those in the r-code. However, a CRC value is not specific to a particular database.

  You can further restrict access to your application database with the DBRSTRCT utility. This procedure restricts everyone's access to the database, including your own. After you run this procedure, no one can write or compile code that updates the database or changes the database's schema. The only code that can access the database is r-code that you compiled before running the DBRSTRCT utility. This level of database protection is available only with r-code. For more information about the DBRSTRCT utility, see Deployment Topics and Tasks on page 169.

Possible drawbacks of r-code

R-code has the following possible drawbacks:

- **Limited portability across user interfaces** — Developer product requirements and r-code portability on page 148 describes the limitations of r-code portability. Depending on the number of incompatible environments requiring a separate compilation, you might have to keep track of multiple r-code trees.
- **Loss of compile-time flexibility** — Because you deliver your application in a precompiled format, you cannot make use of compile-time functionality. For example, you cannot pass arguments to include files to determine sort order, field lists, etc. Users therefore cannot specify their preferences before compilation.
- **More to manage at the development site** — R-code's limited portability often requires you to manage multiple code trees for a single version of a single application. As you modify or fix your application and create new versions, the number of code trees can become quite large.

  R-code is also tightly coupled with the application database. This tight coupling requires you to keep a copy of all of your application databases that have different CRC values. After numerous upgrades and fixes, the number of databases can grow quite large.

9

Initial Deployment

This chapter provides an overview of the steps that are required to deploy an OpenEdge® application using different code formats. It does not cover every possible contingency, but rather tries to give you a general understanding of the initial deployment process for the different code formats. This chapter puts more emphasis on what to do, rather than how to do it.

These steps are not meant to be performed in a strictly chronological order. More detailed information about particular aspects of deployment is provided in Deployment Topics and Tasks on page 169.

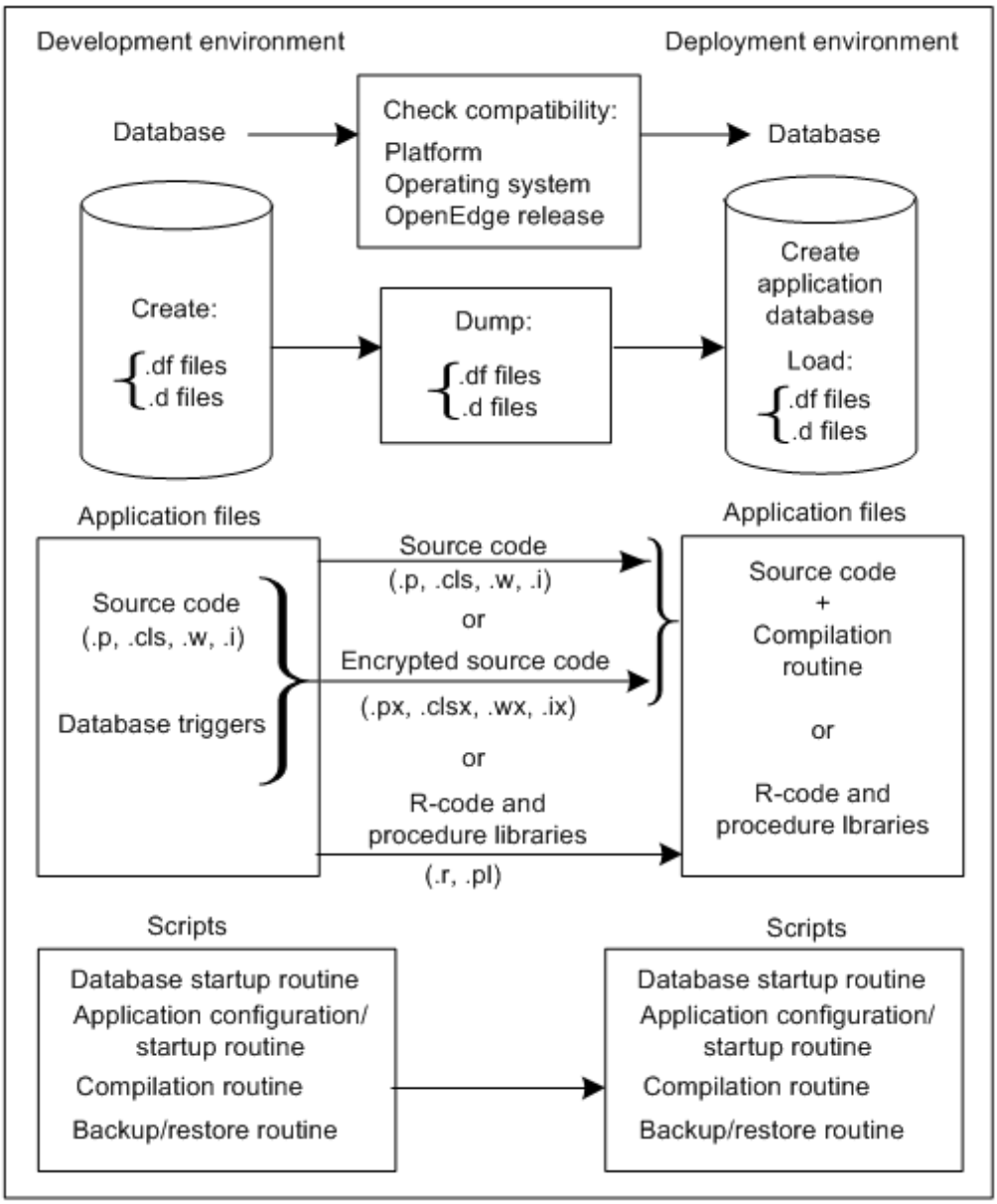
For details, see the following topics:

- Deploying database structures and application files
- Unencrypted source
- Encrypted source
- CRC r-code
- Deploying OpenEdge GUI for .NET applications

Deploying database structures and application files

The following figure illustrates the steps required and options available when deploying new database structures and application files. When deploying a database, Progress Software Corporation recommends that you first create an empty database at the deployment site, then load the `.df` files to create the database structure. This method of deployment will help you avoid any difficulties resulting from incompatibility between platforms, operating systems, or software versions at the development and deployment sites.

Figure 11: Deploying database structures and application files



When deploying application files, you can choose to deploy either unencrypted source code, encrypted source code, or r-code. An important consideration in which method you choose is whether there is a full development product (4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, or WebSpeed Workshop) capable of compiling code at the deployment site. Users with OpenEdge products that cannot compile unencrypted source code will have to deploy encrypted source code and compile it at the site, or deploy r-code. If you deploy r-code, you must ensure that the platforms, operating systems, and software versions at the deployment site match those at the development site. Otherwise, you will have to compile the r-code at the deployment site to ensure compatibility. The following sections address these considerations in more detail.

Unencrypted source

This section provides an overview of the steps to follow to deploy an application using unencrypted source when you are done developing and testing your application.

1. **Dump .df and .d files** — You must dump your application schema (.df) and/or data (.d) into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable.

   Note: Keep in mind that the `.d` and `.df` files should be represented with the appropriate code page for the target platform.

2. **Transfer files** — You must transfer the resultant `.d` and `.df` files and your unencrypted source procedures (which also must be in the appropriate code page) to the target platform. This is a preliminary step to creating your distribution media.

   Note: There is more than one way to transfer files. For example, you can use FTP or other protocols, or you can put the files on CD or other storage media and physically transport the files.

3. **Rebuild the application database at the user site** — You must provide a way for users to build the application database at their site. To do this, the users must be able to load your .df and .d files into an empty database.

   Because you are deploying unencrypted source, users must have a full development product. These products contain a complete Data Dictionary, which users can use to load the `.df` and `.d` files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool.)

   However, even though the user can perform these tasks, usually it is a good idea to write a script that automates this process. This not only frees the user from performing this task, but also makes your application easier to install. A full development OpenEdge product includes all of the Data Dictionary procedures in the *install-dir/gui|tty* subdirectories. The procedures that load data definitions and data are called `load_df.p` and `load_d.p` respectively. Within your script, you can start OpenEdge and run these procedures to load your data definitions and data.

   If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide the users with adequate documentation to guide them through the process.

4. **Copy an empty database and rename it** — To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the installation directory at the user's site. Because databases are not portable, you must use an empty database that already is ported to the target platform.

Users can copy their empty database and rename it, or you can write a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. OpenEdge also provides a newdb script that creates a database. For more information about this script, see Deployment Utilities and Scripts on page 217.

If you decide to let the users perform this step, you must provide adequate documentation to guide them through the process.

5. **Provide an install utility** — You must provide a way for users to install your application. OpenEdge must be installed before they can use your application. You can create one install for both OpenEdge and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the OpenEdge Install batch mode from within it.
6. **Provide a compilation routine** — This step is optional because users must have a full development product, which can compile your application on the fly. However, an application that is precompiled runs faster than an application that is compiled at run time. Therefore, you might want to provide a routine that compiles all of your procedures before it is run.

   You can provide a script to do the compilations, or you can provide the user with adequate documentation to do this task on their own. OpenEdge provides a template procedure, `compile.p`, that you can use to compile your application.
7. **Provide a startup routine** — Provide a way for users to start the application. You can either write a script to do this or provide adequate documentation.
8. **Provide documentation** — You must provide users with enough documentation to use your application and to complete the previous tasks if you do not automate them.
9. **Provide online help** — You might want to provide online help for your application.
10. **Prepare distribution media** — You must prepare your distribution media. It should include all:
    - Procedures and include files
    - `.df` and `.d` files
    - Scripts
    - Bitmap files, if any
    - DLLs, if deploying in MS-Windows
    - Help files
    - Online documentation

    Now you are ready to deploy your application.

Encrypted source

This section provides an overview of the steps to follow to deploy an application using encrypted source when you are done developing and testing your application.

1. **Encrypt your procedures and include files** — Before you can deploy your application, you must encrypt all of your procedures and include files with the XCODE utility. For more information about how to do this, see Encrypting source code procedures on page 169 in in Deployment Topics and Tasks on page 169. Before encrypting the files, make sure they are represented with the appropriate code page for the target platform.
2. **Dump .df and .d files** — As with unencrypted source, you must dump your application schema (`.df`) and/or data (`.d`) files into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable. You do not have to encrypt your `.df` and `.d` files.

   Like your procedure files and include files, the `.d` and `.df` files must be represented with the appropriate code page for the target platform.
3. **Transfer text files** — You must transfer the resultant `.d` and `.df` files to the target platform. This is a preliminary step to creating your distribution media.

   There is more than one way to transfer files. For example, you can use FTP or other protocols, or you can put the files on CD or other storage media and physically transport the files.
4. **Transfer binary files (encrypted files)** — After you encrypt procedures and include files, they become binary files.
5. **Rebuild the application database at the user site** — You must provide a way for users to build the application database at their site. To do this, users must be able to load your `.df` and `.d` files into an empty database.

   Any OpenEdge client, server, or stand-alone database product can compile and run encrypted source. However, only those products that can fully compile can load `.d` files into a database. The 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, and WebSpeed Workshop are the only products with this capability.

   If you deploy to users whose licensed products do not have this capability, you must provide a way for them to load your `.d` files. For more information about providing a dump/load facility for users, see Dumping and loading on page 177.

Note: The dump/load procedures should be considered part of your application. The decision to distribute encrypted source versions of your application procedures applies to the dump/load procedures as well.

A full development product contains a complete Data Dictionary, which users can use to load the `.df` and `.d` files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool. Progress Developer Studio for OpenEdge provides the Database Navigator.) However, even though the user can perform these tasks, usually it is a good idea to write a script that automates this process. This not only frees the user from performing this task, but also makes your application easier to use.

Within your script, you can use pre-existing procedures in the *install-dir*/tty and *install-dir*/gui directories (installed for full development OpenEdge products). Within these directories are procedures that load `.df` and `.d` files, called load_df.p and load_d.p, respectively. Your script can start OpenEdge and run these procedures to load your data definitions and data. To run these procedures at the user site in the case where users do not have a license for a full development product, the procedures must be encrypted or in r-code format. This step is necessary because OpenEdge must be started with the `-rx` parameter to run or compile the procedures where users do not have a license. When started with this startup parameter, OpenEdge cannot run or compile unencrypted source.

If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide adequate documentation to guide them through the process.

6. **Copy an empty database and rename it** — To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the installation directory of all OpenEdge client, server, or stand-alone database products. Because databases are not portable, you must use an empty database that already is ported to the target platform.

   Users can use OpenEdge to copy the empty database and rename it, or you can write a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. OpenEdge also provides a newdb script that creates a database.

   If users have an OpenEdge product that provide these capabilities, and you decide to let them perform this step, you must provide them with adequate documentation to guide them through the process.

7. **Provide an install utility** — You must provide a way for users to install your application. However, OpenEdge must be installed before they can use your application. You can create one install for both OpenEdge and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the OpenEdge Install batch mode from within it.

8. **Provide a compilation routine** — When you deploy with encrypted source, you typically provide a compilation routine. Though this is not necessary in all cases, it is usually a good idea because it frees the user from performing the task of compiling your application. In the case where you specify unique encryption keys, providing a compilation routine becomes a necessity.

   All OpenEdge client, server, and stand-alone database products can compile encrypted source. Therefore, if you use the default encryption key, users can compile your application without a specialized routine. However, if you have users compile your application, you must provide adequate documentation to guide them through the process.

   Because encrypted code is not precompiled, your application can take advantage of compile-time functionality. For example, you might want to pass arguments to include files to determine sort order, field lists, and so on. You can take advantage of this behavior at the user site, allowing users to specify their preferences before compilation. This option is not available with r-code, which must be compiled at the development site. Whether you can repeatedly take advantage of compile-time features during the ordinary day-to-day operation of your application depends on what OpenEdge products the users have.

   To compile encrypted source code, OpenEdge must be started with the `-rx` startup parameter. When an OpenEdge session is running with this startup parameter, ABL (Advanced Business Language) can compile encrypted source but cannot compile unencrypted source. If users have a full development product, they will probably want to write and compile their own unencrypted procedures. Thus, in their day-to-day operation of OpenEdge, they must run OpenEdge **without** the `-rx` startup parameter. For users with full development products, therefore, you probably want to perform a one-time compilation with the `-rx` parameter, then restart OpenEdge without this parameter. For users with OpenEdge products that cannot compile ordinary unencrypted source, running permanently with the `-rx` parameter is not an issue.

   OpenEdge provides a template procedure, `compile.p`, that you can use to compile your application. For more information about `compile.p`, see Compilation on page 171.

9. **Provide a startup routine** — You must provide a way for users to start the application. You can either write a script to do this or provide adequate documentation.

10. **Provide documentation** — You must provide users with enough documentation to use your application and to perform the previous tasks if you do not automate them.

11. **Provide online help** — You might want to provide online help for your application.

12. **Prepare distribution media** — You must prepare your distribution media. It should include all:

    - Procedures and include files (encrypted)
    - Data definition (`.df`) and data (`.d`) files
    - Scripts

- Bitmap files, if any
- DLLs, if deploying on MS-Windows
- Help files
- Online documentation

Now you are ready to deploy your application.

CRC r-code

This section provides an overview of the steps to follow to deploy an application using CRC-based r-code when you are done developing and testing your application. For a complete description of CRC-based r-code, see R-code Features and Functions on page 189.

To deploy your application:

1. **Decide whether to restrict database access** — If you deploy your application to users with full development products, they can write their own applications to access the database and modify the database schema. Depending on the nature of your application, you might not want this to happen. To restrict access to the database, you can use the DBRSTRCT utility that comes with OpenEdge. For more information about using this procedure, see Restricting database access on page 179.

   If you decide to restrict access to the database, you must physically deploy a copy of your application database with your application. It is not possible for users to build your database at their site. Because databases are not portable, you must generate a copy of the database for each target platform. Then you must run the DBRSTRCT utility on each platform against all of the databases.

   In general, restricting access to the database makes it more difficult to initially deploy and upgrade applications, but it also helps minimize user-created errors. For each release of the application, the development site must:

   - Generate a database for each platform
   - Compile the application against each database
   - Save copies of each database and each r-code tree

   To update the application, the user must dump the existing database and load the data into a new database that you supply. For users with large mission-critical databases, this might be an unacceptable requirement.

2. **Compile your procedures and include files** — As described in Choosing a Code Format on page 145 r-code is not completely portable. For each combination of factors requiring a separate compilation, you need to perform a separate compilation and keep track of a different code tree at the development site. For each combination, you also need a full development OpenEdge product to do the compilations.
3. **Dump .df and .d files** — If you decide not to restrict database access, you should dump your application schema (`.df`) and/or data (`.d`) files into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable. The `.d` and `.df` files should be represented with the appropriate code page for the target platform.
4. **Transfer text files** — You must transfer the resultant `.d` and `.df` files to the target platform.

   There are several ways to transfer files. For example, you can use FTP or other protocols, or you can put the files on CD or other storage media and physically transport the files.

5. **Transfer r-code files** — To aid in moving r-code files between platforms, you can use the BUNDLE utility. For more information about the BUNDLE utility, see Deployment Topics and Tasks on page 169.

6. **Rebuild the application database at the user site** — If you have decided not to restrict access to the database, then you must provide a way for users to rebuild your application database at their site. To do this, users must be able to load your `.df` and `.d` files into an empty database.

   Note: You can restrict access to the database and provide compiled code to load `.d` files.

   Any OpenEdge client, server, or stand-alone database product can run r-code. However, not all of these products can load `.d` files into a database. Full development products are the only products that have this capability.

   If you deploy to users whose licensed products do not have this capability, you must provide a way for them to load your `.d` files. For more information about providing a dump/load facility for users, see Dumping and loading on page 177.

   Note: The dump/load procedures should be considered part of your application. The decision to distribute r-code versions of your application procedures applies to the dump/load procedures as well.

   Full development products contain a complete Data Dictionary, which users can use to load the `.df` and `.d` files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool. Progress Developer Studio for OpenEdge provides the Database Navigator.) However, even though the user can perform these tasks, usually it is a good idea to write a script that automates this process. This not only frees the user from performing this task, but also makes your application easier to use.

   Within your script, you can use pre-existing procedures in the *install-dir*`/tty` and *install-dir*`/gui` directories (installed for full development OpenEdge products). Within these directories are procedures that load `.df` and `.d` files, called `load_df.p` and `load_d.p`, respectively. Your script can start OpenEdge and run these procedures to load your data definitions and data. To run these procedures at user sites where users do not have a license for the 4GL Development System or OpenEdge, the procedures must be in r-code format.

   If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide adequate documentation to guide them through the process.

7. **Copy an empty database and rename it** — To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the installation directory of all OpenEdge client, server, or stand-alone database products. Because databases are not portable, you must use an empty database that already is ported to the target platform.

   Users can use OpenEdge to copy the empty database and rename it, or you can free them from this responsibility by writing a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. OpenEdge also provides a newdb script that creates a database.

   If users have an OpenEdge product that provide these capabilities, and you decide to let them perform this step, you must provide adequate documentation to guide them through the process. For example, you should let them know there are multiple empty databases, and you should recommend which one they should use.

8. **Provide an install utility** — You must provide a way for the user to install your application. OpenEdge must be installed before they can use your application. You can create one install for both OpenEdge and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the OpenEdge Install batch mode from within it.
9. **Provide a startup routine** — You must provide a way for the user to start the application. You can write a script to do this or provide adequate documentation.

10. **Provide documentation** — You must provide the user with enough documentation to use your application and to perform the previous tasks if you do not automate them.
11. **Provide online help** — You might want to provide online help for your application.
12. **Prepare distribution media** — You must prepare your distribution media. It should include all:
    - Compiled procedures
    - Application procedures
    - Dump and load procedures
    - Data definition (`.df`) and data (`.d`) files
    - Scripts and bitmap files, if any
    - DLLs, if deploying in MS-Windows.
    - Help files
    - Online documentation

Deploying OpenEdge GUI for .NET applications

This section discusses the extra considerations for deploying an OpenEdge GUI for .NET application. In general, you perform the same tasks in the same order and have the same options for the kind of source code you deploy. The .NET portion of a GUI for .NET application requires additional configuration files and directory structures.

Installing .NET Framework

Any client machine running the GUI for .NET requires .NET Framework 3.0 or later. When deploying a GUI for .NET application, you must check for an appropriate version and install the framework if necessary. The OpenEdge installer has an option to install .NET Framework 3.0 if it is missing. If you create your own custom installation package for a GUI for .NET application, it must check for and, if necessary, install an appropriate version of the framework before it installs OpenEdge.

Note: Microsoft requires users to accept an EULA before installing the .NET Framework. The Microsoft EULA appears during the OpenEdge installation if the framework is installed. This prevents silent installs on client machines that do not already have an appropriate version of the framework.

If your application makes extensive use of the GUI for .NET, consider using the Preload CLR (`-preloadCLR`) startup parameter to force the .NET CLR and any specified assemblies to load at startup. For more information on `-preloadCLR`, see *Startup Command and Parameter Reference*.

Creating an assemblies directory

An *assembly* is a .NET library or collection of classes. By default, .NET would look for all assemblies used by a GUI for .NET application in the `OpenEdge-install-dir/bin` directory. However, your applications should not install files in the `OpenEdge-install-dir/bin` directory.

Instead, GUI for .NET applications have an assemblies directory that contains the assemblies and the related configuration files. By default, the assemblies directory is the session's working directory. However, you can use the Assemblies (`-assemblies`) startup parameter to specify a different assemblies directory. For more information on `-assemblies`, see *Startup Command and Parameter Reference*.

Deploying application source code

As in other ABL applications, your OpenEdge session uses the PROPATH to look for the ABL source code for a GUI for .NET application. A GUI for .NET application includes additional types of configuration and source code files to support the .NET controls. OpenEdge looks for these files in the assemblies directory.

The additional source code files include:

- **OpenEdge internal assemblies** — OpenEdge uses several assemblies to connect the ABL and .NET sides of the application. OpenEdge installs these assemblies in the `OpenEdge-install-dir/bin` directory.
- **OpenEdge Ultra Controls for .NET** — The GUI for .NET includes support for a set of .NET controls. The Ultra Controls are licensed separately for development environments. However, OpenEdge deployment products install assemblies containing run-time versions of these controls that do not require a separate license. OpenEdge installs these assemblies in the `OpenEdge-install-dir/bin` directory and looks for them there.
- **ABL controls derived from .NET controls** — OpenEdge handles an ABL class that inherits from a .NET class in the same way as every other ABL class. The application PROPATH must contain the class file or its r-code (`.r` or `.pl`).
- **Application-specific assemblies** — Application-specific assemblies might include third-party controls or your own custom .NET assemblies. You can install these assemblies in the .NET Global Assembly Cache (GAC) or in the application's assemblies directory.
- **.NET Resource Files** — Some .NET controls require resource files (`.resx`) to operate. Deploy resource files to the same directory as the assemblies containing the controls that require the resource file.
- **.NET Icons and Bitmaps** — If a .NET control uses .NET icons (`.ico`) and bitmaps (`.bmp`), the icons and bitmaps generally should be in the same directory as the code for the controls that use them.

The additional configuration files include:

- **Assembly References file** — The Assembly References file (`assemblies.xml`) lists all the assemblies used by a GUI for .NET application. You must deploy this file in the application's assemblies directory. For more information on `assemblies.xml`, see .NET Framework on page 81.
- **Assemblies Application Configuration file** — An Assemblies Application Configuration file (`assemblies.config`) is a means to alter how specific assemblies are loaded in exceptional situations. You must deploy this file in the application's assemblies directory. For more information on `assemblies.config`, see .NET assemblies on page 81.
- **Application Configuration file** — OpenEdge might require an Application Configuration file (`prow32.exe.config`) to configure its internal assemblies. If one is needed, OpenEdge provides the file and installs it in `OpenEdge-install-dir/bin`.

Note: You should never use or modify this file directly.

10

Upgrades

After you distribute your application and users have been using it, you might want to upgrade, fix, or modify either the application procedures or the application database. It is a good idea to develop a strategy for dealing with application upgrades before distributing an application. That way, when you have to upgrade an application, you are less likely to have unexpected problems.

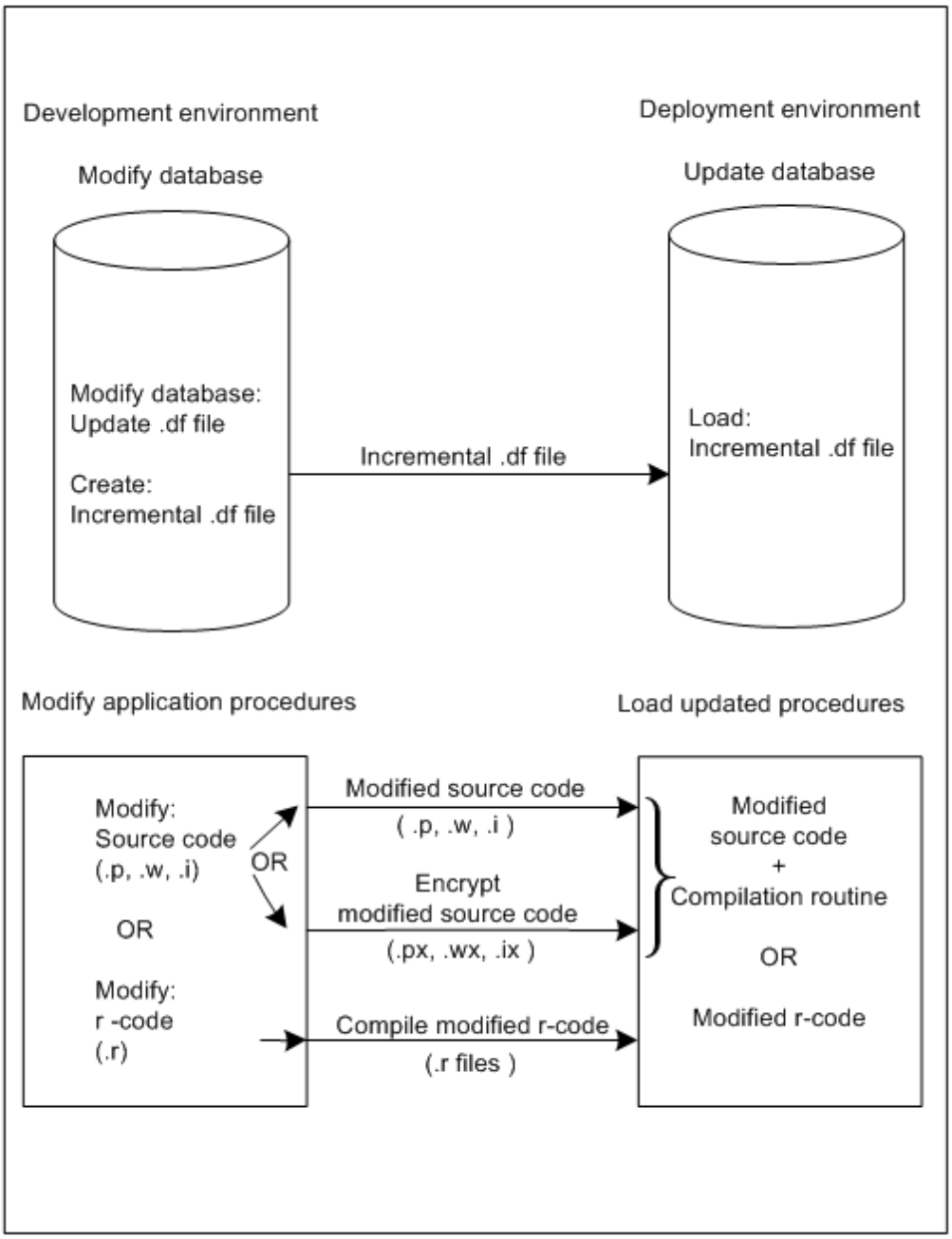
For details, see the following topics:

- Deploying upgrades
- Modifying the application procedures
- Modifying the application database

Deploying upgrades

The following image illustrates the steps needed and options available when upgrading and distributing changes to the database structure or application procedures. When making changes to the database structure, you must create an incremental `.df` file that contains the changes in the database structure. You can then load this `.df` file into the existing database structure to update that database. For information on creating an incremental ABL (Advanced Business Language) `.df` file, see *Manage the OpenEdge Database*. See R-code Features and Functions on page 189 for more information about deploying changes to database structures.

Figure 12: Deploying upgraded database structure and application procedures



When modifying application procedures, you can choose to deploy either unencrypted or encrypted source code, or to deploy modified r-code. An important consideration for which method you choose is whether there is a full development product (4GL Development System, OpenEdge Studio, and Progress Developer Studio for OpenEdge, or WebSpeed Workshop) capable of compiling code at the deployment site. When deploying to users with OpenEdge products that cannot compile unencrypted source code, developers will have to deploy encrypted source code and compile it at the site, or deploy r-code. If you deploy r-code, you must ensure that the platforms, operating systems, and software versions at the deployment site match those at the development site. Otherwise, you will have to compile the r-code at the deployment site to ensure compatibility.

Modifying an application is relatively straightforward if you have not made changes to the database schema. If you have made changes to the database schema, modifying the application requires more steps.

Modifying the application procedures

As with any distribution, you can choose to distribute unencrypted source, encrypted source, or r-code. This section describes upgrading an application without any modifications to the database. If you have modified the database schema, upgrading becomes more complicated, as described in Modifying the application database on page 163.

Unencrypted source

To upgrade your application, distribute the modified versions of your procedures.

Encrypted source files

To upgrade your application, distribute the modified versions of your procedures. If you specify an encryption key, you must also provide a compilation procedure.

As described in Initial Deployment on page 151 when you deploy with encrypted source, you usually want to provide a compilation procedure. Although this is not necessary in all cases, it is a good idea because it frees the user from having to compile your application. When you specify unique encryption keys, providing a compilation routine is a necessity.

All OpenEdge client, server, and stand-alone database products can compile encrypted source. Therefore, if you use the default encryption key, the user can compile your application without a specialized procedure. However, if you have the user compile your application without a script, you must provide adequate documentation to guide them through the process.

CRC-based r-code

If you supply new or modified CRC-based r-code files, you must precompile the procedures against a version of the database that is structurally the same as the user's database. You must also compile it on a machine of the same class, same user interface, and same combinations of DataServers (if any). It does not matter when the database tables were created or modified.

Modifying the application database

Upgrading is more complicated when you have to change the database schema. If you are upgrading with unencrypted source, encrypted source, or CRC-based r-code, you can supply an incremental `.df` file and copies of the modified procedures.

Upgrading with updated .df and encrypted source

When you update a database structure in the development environment, you must then re-create these changes in the database that is already established at the deployment site. To accomplish this transfer, an incremental `.df` file is generated in the development environment. Copies of the original and updated databases are compared, and the differences between them are incorporated into an incremental `.df` file. For more information on creating an incremental ABL data definitions file see *Manage the OpenEdge Database*.

The following general steps tell you how to prepare a `.df` file that includes new or modified data definitions:

- Obtain copies of the current user database and the new modified database.
- Connect the two databases.
- Compare the data definitions in the user database with the data definitions of your new updated database, creating a new incremental `.df` file that reflects the changes.
- Provide a utility for users that loads the incremental `.df` file into the existing database.
- Encrypt the procedures affected by changed data definitions.

To prepare a data definitions file:

1. In Windows, open the **Data Administration** Tool (for character interfaces, open the **Data Dictionary**).
2. Connect the two databases by choosing **Database** > **Connect**.
3. Select the database that includes the new, modified data definitions as your current database by choosing **Database** > **Select Working Database**.
4. Create an incremental definition file by choosing **Admin** > **Dump Data and Definitions** > **Create Incremental .df File**.

   This option compares the data definitions in the non-empty copy to the current database schema and creates a new data definition (`.df`) file. The new `.df` file contains a record for each difference between the two schemas. The differences include any added, renamed, changed, or deleted file, field, or index.

   If a file, field, or index exists in the old database but not in the new schema, OpenEdge asks you whether the object has been renamed. If you respond `no`, a record appears in the new `.df` file marking the object as deleted.

   If the new schema includes a new unique active index, OpenEdge asks you whether you want to deactivate it. If you do not deactivate the index, and there are duplicate keys in the old database, OpenEdge aborts your attempt to load new definitions into the old database. If you deactivate the index, the load procedure defines the new index, but does not create the index file. You must build and activate the index after loading the new data definitions.

5. Perform steps a through d below for testing purposes. Then prepare a tool for users that performs these steps on site:

   a) Select the copy of the old database as your working database.

   b) Load the updated data definitions by choosing **Admin** > **Load Data and Definitions** > **Load Data Definitions (.df file)**.

   c) If you deactivated any indexes in Step 3, re-create data in the indexed fields as required to avoid duplicate keys. Then reactivate the indexes with `PROUTIL IDXBUILD`. For more information, see *Manage the OpenEdge Database*.

   d) OpenEdge now updates the old database schema to match the modified schema. Compile and test all your procedures against the updated database.

Note: The upgrade template provided with OpenEdge outlines one way to do this. Before you can use this template, you probably need to modify it. See Using the upgrade template on page 165.

6. Test your procedures.
7. Use the XCODE utility to encrypt procedures invalidated by the new data definitions.

   Since the new .df file changes CRC check sums and time-stamps only on the database tables that are actually modified, you only have to encrypt and ship the source versions of those procedures that access the changed tables. (If you compile your procedures with the XREF option, you get a listing of the tables accessed by each procedure.)

Using the upgrade template

The upgrade template, all .p files it uses, and all .inp files it uses are only templates; you must check each file and modify it to suit your particular application. Following is a list of checks and modifications you must make before using upgrade; your application might require additional changes.

The UNIX version of upgrade is found in */install-dir*/toolkit/samples/unix. The DOS version is located in */install-dir*/toolkit/samples/dos. The files that upgrade uses can be found in */install-dir*/toolkit/samples.

The upgrade template performs three basic actions to upgrade an application at the user's site:

1. Loads the new .df file, which you created from your version of the upgraded database with the **Admin** > **Dump Data and Definitions** > **Create Incremental .df File option**
2. Rebuilds any indexes deactivated during the data definition load process
3. Compiles encrypted source procedures

   The changes to the upgrade template for the data definition load stage (Step 1) include modifying upgrade.p (not the upgrade template) and the files it uses:

   - Set the system administrator ID and password appropriately in loginupg.p (or prepare a tool your users can use to set them).
   - Modify upgrade.inp to name the new .df file.
   - Test load_df.p (a standard OpenEdge Dictionary procedure). If you created a new unique active index, load_df.p asks if you want to deactivate the index. You need to add a response (Y or N) to upgrade.inp.
   - If you and your users do not freeze the database files, delete the call to freeze.p. (Leave the call to unfreeze.p in case your users ever freeze the database files without your knowledge.)
   - Modify unfreeze.p and freeze.p to include the correct filenames.
   - Check upgrade.log for any problems.

   The changes to the upgrade template for the rebuild deactivated indexes stage (Step 2) are:

   - If you and your users will not be deactivating indexes, remove the _proutil command line from upgrade.
   - If you added a new unique active index, modify rebuild.inp to include the correct responses to prompts by _proutil $1 -C idxbuild. That is, test _proutil idxbuild on your database and add a line to rebuild.inp to answer each question appropriately.

- Check `rebuild.log` for problems.
- You might want to add a procedure to `upgrade` that checks `rebuild.log` and, if necessary, fixes duplicate key data and rebuilds the index again. Alternatively, you might want to check the `.log` file and just stop before compilation proceeds.

The changes to `upgrade` template for the compile encrypted procedures stage (Step 3) are:

- Modify `compile.inp` to name the procedures to be compiled.
- Modify the `COMPILE` statement in `compile.p` to use the encryption key you want.
- Prepare to ship `compile.p`: either encrypt `compile.p`, possibly with the default key, or compile `compile.p`, thereby hiding your encryption key in the `.r` file. If you ship `compile.r`, it must be compiled on the target machine type. It can be compiled against any database because it does not reference any database files. The `upgrade` template is relevant when you want to distribute only new data definitions and encrypted versions of affected application procedures. However, you might choose to ship a complete new version of the database and application instead, as described in the following section.

Upgrading with updated .df and CRC-based r-code

The flexibility of CRC-based r-code makes it easy to distribute a new release of an existing application.

To distribute a new release of an existing application:

1. Create an incremental `.df` file of your application database using the **Data Administration** tool (for graphical environments) or the **Data Dictionary**'s **Admin** submenu (for character environments). For more information on creating an incremental `.df` file, see *Manage the OpenEdge Database*.
2. Recompile source files that were affected by the new schema changes.
3. Send the `.df` file and copies of the new r-code files that were affected by the schema changes to your users. After the users load the incremental `.df` file, they can immediately run your new `.r` code.

   Essentially, the steps you take are the same as when you use encrypted source, but you gain the following advantages:

   - You do not have to encrypt your procedures. However, you need to recompile some r-code.
   - The user does not have to compile your procedures.

   One possible disadvantage of using CRC-based r-code is that a user with the 4GL Development System, OpenEdge Studio, and Progress Developer Studio for OpenEdge, or WebSpeed Workshop can create a counterfeit database (a database with the same structure as your application database). The user can then write, compile, and run it against your application database.

   If this concerns you, you can either turn on run-time permission checking or use the PROUTIL utility's `DBAUTHKEY` qualifier. For details on how to turn on run-time permission checking, see Maintaining Application Security on page 39.

   The PROUTIL utility's `DBAUTHKEY` qualifier enables you to set an authorization key for your database. When compiling a procedure, OpenEdge includes the value of this key in your r-code. CRC-based r-code that does not include the correct authorization key will not run against your database. You can insert the authorization key into existing CRC-based r-code with the PROUTIL utility's `RCODEKEY` qualifier. For more information on the PROUTIL utility's `DBAUTHKEY` and `RCODEKEY` qualifiers, see *Manage the OpenEdge Database*.

OpenEdge product upgrades

The user might upgrade from an OpenEdge product that does not support full development to either the 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, or WebSpeed Workshop.

To allow the users of a previously restricted application database to run your application using the capabilities of their new, less restrictive version of OpenEdge:

1. Create a new basic database.
2. Distribute the new basic database to the user.

   The user must use the dump/load facility you provided to dump the existing database, make a copy of the new basic database, and reload the existing data into that new database.

   If the new version of OpenEdge is being installed in a different directory than the old version, UNIX users can use the `_tlr` module to edit the value of the `DLC` variable in your application scripts.

11

Deployment Topics and Tasks

This chapter describes issues related to deploying OpenEdge® application.

For details, see the following topics:

- Encrypting source code procedures
- Using the BUNDLE utility
- Dumping and loading
- Restricting database access

Encrypting source code procedures

This section describes the XCODE utility and the process of encrypting source code before deploying an application.

XCODE utility

Use the XCODE utility to encrypt ABL (Advanced Business Language) procedures and include files. This is the syntax:

Syntax (UNIX)

```
xcode [ -k key] -d directory[files] [ - ]
```

key

A character string up to eight characters long. On UNIX, *key* is case sensitive. If you do not specify *key*, XCODE uses a default key. When OpenEdge is started with the Encrypted Compiler Mode (-rx) startup parameter, ABL automatically uses this default key, unless you specify the XCODE option on the COMILE statement or set the XCODE-SESSION-KEY attribute on the SESSION-POLICY handle. (The XCODE option takes precedence over the XCODE-SESSION-KEY attribute if both are in effect for a compile operation.

The XCODE utility uses the default codepage of the operating system where it runs. To avoid possible conflicts from automatic codepage conversions, key values should use only US-ASCII characters. These characters are included in all codepages and can be located below code point 128 in the codepage.

Note that keys longer than eight characters do not generate warning or compile errors. Both the XCODE utility and the COMPILE statement ignore extra characters if present.

directory

The directory where XCODE places the encrypted files. The specified directory must be different from the source directory, because each encrypted file retains the name of its original source file. XCODE overwrites existing files in directory if they are encrypted.

files

The pathnames, relative to the current directory, of the files you want to encrypt. XCODE appends the pathnames supplied to *directory*.

Preparing to use XCODE

Before running XCODE, create a directory structure exactly parallel to your source directory structure, then move to the root of the original source directory structure. Supply the root of the new structure as *directory*, and then supply the pathnames, relative to the current directory, of the files to be encrypted as *files*. The relative path names are appended to *directory*. Because XCODE does not create new directories, any directories implied by the *directory* or *files* arguments must be created before running XCODE. Encrypted files have the same names as the original source files.You must encrypt both your procedure and include files in order for your encrypted procedures to run.

The dash (-) option tells XCODE to take filenames from the standard input. On UNIX, this allows you pipe output from an ls, cat, or find command to XCODE.

The following shows how you can use XCODE on UNIX. Suppose you have just finished developing and testing the procedures in /usr/test and its four subdirectories and you want to encrypt them for shipment to users.

To encrypt procedures for shipment to users:

1. Create a directory structure parallel to `/usr/test`, as shown:

```
cd /usr/test
mkdir /usr/testx
mkdir /usr/testx/ar
mkdir /usr/testx/fm
mkdir /usr/testx/inv
mkdir /usr/testx/oeFk
```

2. Move to the original development root in preparation for encryption, as shown:

```
cd /usr/test
```

3. Encrypt your procedures, as shown:

```
xcode -k mykey -d usr/testx *.p ar/*.p fm/*.p inv/*.p oe/*.p
```

   Encrypted versions of the source procedures in `/usr/test` and its subdirectories now reside, with the same names, in `/usr/testx` and its subdirectories.

   Note: You must also encrypt include (`.i`) files.

Compilation

If you distribute encrypted source code, you must also prepare and distribute a program that compiles your application procedures at a user site. The core of this program is one or more `COMPILE` statements with the `XCODE` option. If you encrypt your procedures using the default key, the `XCODE` option is unnecessary because the Compiler recognizes that the procedures are encrypted.

The `XCODE` option of the `COMPILE` statement compiles the specified source files, and any encrypted include files, using the same keys supplied in the `XCODE` option.

Alternatively, you can specify a key at the session level. The `XCODE-SESSION-KEY` attribute of the `SECURITY-POLICY` handle, when set, overrides the default key but does not override the `XCODE` option on a `COMPILE` statement.

When started with the Encrypted Compiler (`-rx`) startup parameter, OpenEdge places you (or a user) by default in the Procedure Editor. The `COMPILE` statement and the `icompile.p` procedure will only compile encrypted procedures when OpenEdge is started with `-rx`. When shipping encrypted source code, you must supply either an encrypted `.p` or a `.r` version of your compile program, which can be run in encrypted compiler mode. The OpenEdge installation includes a sample template, called `upgrade`, that invokes OpenEdge with the `-rx` parameter and compiles encrypted source programs.

You invoke the encrypted source code compiler with the command `PROGRESS/XCOMPILER`. See `upgrade.com` in the Toolkit `samples` subdirectory for more information.

After the procedures are compiled, you might want users to delete the source to regain the disk space. If you do this, consider freezing the database files; otherwise, the user can modify the database and invalidate object procedures for which the user no longer has the source.

Using the BUNDLE utility

The BUNDLE utility lets you put binary files into a file called a bundle, transport the bundle to another machine by one of the standard means, and extract the files from the bundle. The BUNDLE utility is designed for copying encrypted source files and r-code files, but any platform-independent binary file or ASCII file can be copied as well. It is not designed to handle executables, object files, OpenEdge RDBMSs, or other platform-dependent files.

The BUNDLE utility consists of two C language programs: `bundle.c` and `unbundle.c`. These programs are described in the following sections.

The bundle.c program

The bundle program puts input files into a bundle. This is the command syntax to run it:

Syntax (UNIX)

```
bundle [ -ascii | -binary ] [ -uppercase | -lowercase ] bundle-file { { -select
[ selection-file ] } | input-file }
```

`-ascii | -binary`

Specifies whether the input file is to be read as an ASCII text file or as a binary file. The default is binary.

`-uppercase | -lowercase`

Tells bundle to convert the names of files being bundled to uppercase or lowercase. By default, filenames are used without case conversion.

`bundle-file`

Provides the name of the file into which input files are bundled. If the file does not exist, bundle creates it. If it does exist, bundle appends data to the end of it.

`-select` *`selection-file`*

Indicates that the names of the input files are to be read from a selection file. By default, the selection file is read from standard input.

This is the Syntax of *`selection-file`*:

```
filename
[ ascii | binary ]
```

Each entry must be on a line by itself and cannot continue across lines.

If ascii or binary is absent, the command line specification is used. Use an exclamation point (!) or pound sign (#) as comment characters. Everything from a comment character to the end of line is ignored. The comment character must be separated from any preceding text by a blank space. Blank lines are also ignored. Wild cards are not supported.This is a sample file:

```
# First line.
# Config files follow.
cfg-show.x   # Depending on command-line, ascii or binary.
cfg-doc.txt  ascii
cfg-make     ascii
# The next line is blank.    # Last line.
```

input-file

The name of a single input file. Wild cards are not supported.

The unbundle.c program

The unbundle program takes files out of a bundle. This is the command syntax to run it:

Syntax (UNIX)

```
unbundle [ -toc | -fulltoc | -date | -all ] bundle-file
             [ -select [selection-file ] ]
```

-toc | -date | -fulltoc | -all

Indicates the operation to perform. The default, toc, lists a table of contents that shows the name and type (ascii or binary) of each file in the bundle. The fulltoc value lists a table of contents containing toc information, date information, and the input files' file specifications, modify dates, and sizes. The date value lists time stamps showing when bundle was run. The all value unbundles all files in the bundle putting the unbundled files in the current directory.

-select *selection-file*

Allows you to select by name the files to unbundle or to list with toc or fulltoc.

This is the Syntax of *selection-file*:

```
filename
                        [ unbundle-as-file-spec ]
```

The *filename* value names the file to be processed. When you are unbundling, unbundle-as-file-spec specifies the name unbundle is to use as the unbundled filename. Comments and blank lines are permitted as described earlier for the bundle command's *selection-file*. Wild cards are not supported.

If select is specified and no *specification-file* is provided, selections are read from standard input.

The following is a sample file:

```
# UNIX example
# First line.
cfg-show.x   pdir/cfg-show.p
cfg-doc.txt  docdir/cfg-doc.doc
cfg-make     # Created as cfg-make in current directory.
readme.txt   # Created as readme.txt in current dir.
# The next line is blank.

# Last line.
```

Examples using BUNDLE

The following synopsis shows how to use bundle and unbundle. This section assumes you have to transfer encrypted source files, which are binary files.

First you have to create a "bundle" file. To do this you specify the output file (that is, the bundle) and one or more input files (the encrypted source files). For these examples, the bundle file has a `.bun` extension and the encrypted source files have a `.x` extension. The bundle is created on UNIX, copied to Windows, and unbundled in Windows.

Put a single file named `foo.x` in the bundle, as shown:

```
# bundle xcode.bun foo.x
```

Or, put all encrypted source files in the current directory in the bundle, as shown:

```
# ls *.x | bundle xcode.bun -select
```

Or, put some selected files (this, that, what) in the bundle, as shown:

```
# bundle xcode.bun -select
this.x
that.x
what.x
^D
```

You can also put the selections into a file, as shown:

```
# cat >xcode.sel
this.x
that.x
what.x
^D
# bundle xcode.bun -select xcode.sel
```

Note that files are always appended to the bundle, so you can incrementally add files to it. Thus, the above sequence created `xcode.bun` with this in it: `foo.x`, all xcode files in the current directory, `this.x`, `that.x`, and `what.x`.

Now you need to copy the bundle to some destination. You can use FTP, making sure that binary mode is in effect.

To see what is in the bundle, enter this command:

```
C:> unbundle /toc xcode.bun
```

The unbundle program lists a short table of contents. To see a full table of contents, enter this command:

```
C:> unbundle /fulltoc xcode.bun
```

To unbundle everything in the bundle and put the files in the current directory, enter this command:

```
C:> unbundle /all xcode.bun
```

To unbundle selected files (`this.x` and `that.x`) and put them in specified directories, enter this command:

```
C:> unbundle xcode.bun /select
this.x   usr2:[xcode]this.x
that.x   usr1:[xcode]that.x
^Z
```

You can also put the selections into a file, as shown:

```
C:> create xcode.sel
this.x   usr2:[xcode]this.x
that.x   usr1:[xcode]that.x
^Z
C:> unbundle xcode.bun /select xcode.sel
```

Once you have unbundled the files, they can be executed or compiled in OpenEdge.

Text files can also be bundled. You can specify `-ascii` on the command line, thereby making ascii the default file type; or you can specify ascii after the filename in the selection list, as shown:

```
C:> dir *.p | bundle -ascii pcode.bun
C:> bundle pcode.bun -select
foo.p ascii
^D
C:>
```

Binary and ascii can be mixed in a bundle, as shown:

```
C:> bundle allcode.bun -select
foo.p ascii
foo.x binary
^D
C:>
```

Since bundle always appends files to the end, multiple files with the same name might appear in a bundle, as shown:

```
C:> bundle allcode.bun -select
foo.p ascii
xcode/foo.p binary
^D
C:>
```

In this example, the filename `foo.p` appears twice and represents two different files. When unbundling this (with -all) `foo.p` (ascii) is unbundled first and then `foo.p` (binary) is unbundled overwriting the other `foo.p`. To unbundle both files, use a selection list to give each file a unique name or put each in a different directory, as shown:

```
C:> unbundle allcode.bun -select
foo.p src/foo.p   # This is the ascii foo.p
foo.p xcode/foo.p # This is the binary foo.p
^D
C:>
```

You can also use a selection list to skip over a file by directing it to the null device, as shown in the following UNIX example:

```
$ unbundle allcode.bun -select
foo.p /dev/null
foo.p
^D
$
```

To help differentiate files with identical names, unbundle -full shows files' input file specifications, modify dates, and sizes in bytes; the current directory when the bundle was created; and a time stamp showing when the files were bundled, as shown:

```
$ unbundle mystuff.bun -full
# V0 Time-stamp: Thu Apr 15 19:59:19 1993
# V0 Current-directory: C:\TEMP
 mon03.txt      #ascii
# V0 Input-file: MON03.TXT
# V0 File-modify-time: Thu Apr 15 19:57:12 1993
# V0 File-size: 2503
 mon03.txt      #ascii
# V0 Input-file: ..\WVM\DOC\MON03.TXT
# V0 File-modify-time: Sun Apr 04 18:17:48 1993
# V0 File-size: 2655
# V0 Time-stamp: Thu Apr 15 20:00:23 1993
# V0 Current-directory: C:\TEMP
 mon03.txt      #ascii
# V0 Input-file: mon03.txt
# V0 File-modify-time: Thu Apr 15 20:00:10 1993
# V0 File-size: 2666
$
```

V0 denotes version 0 of the bundle record format. It allows backward compatibility should this format have to be changed. You can use `-select` with `-full` to see information on selected file names.

Note: Binary dump and loads are dramatically faster. See the section on PROUTIL in *Manage the OpenEdge Database* for more information on binary dump and load.

Dumping and loading

You have to supply a facility to dump and reload the application database. Your users need a dump and reload facility to:

- Convert data to ASCII format for transporting among machines or database products
- Increase index efficiency
- Convert from one version of the software to another

The 4GL Development System, OpenEdge Studio, Progress Developer Studio for OpenEdge, and WebSpeed Workshop products all provide full dump/load capabilities via either the GUI Data Administration Tool, Character Data Dictionary Tool, or command line PROUTIL utility. For more information about these capabilities, see *Manage the OpenEdge Database*. Other OpenEdge products provide less complete dump/load capabilities. Other OpenEdge products such as Query Results provide less complete dump/load capabilities.

Note: Progress Software Corporation recommends you use binary dump and load instead of MKDUMP when moving table contents. Refer to *Manage the OpenEdge Database* for more information about binary dump and load, and loading of sequence values.

The Progress Developer's Toolkit includes sample utilities that you can use to create your own dump/load facility. This section covers these samples and explains how to modify them for your own use.

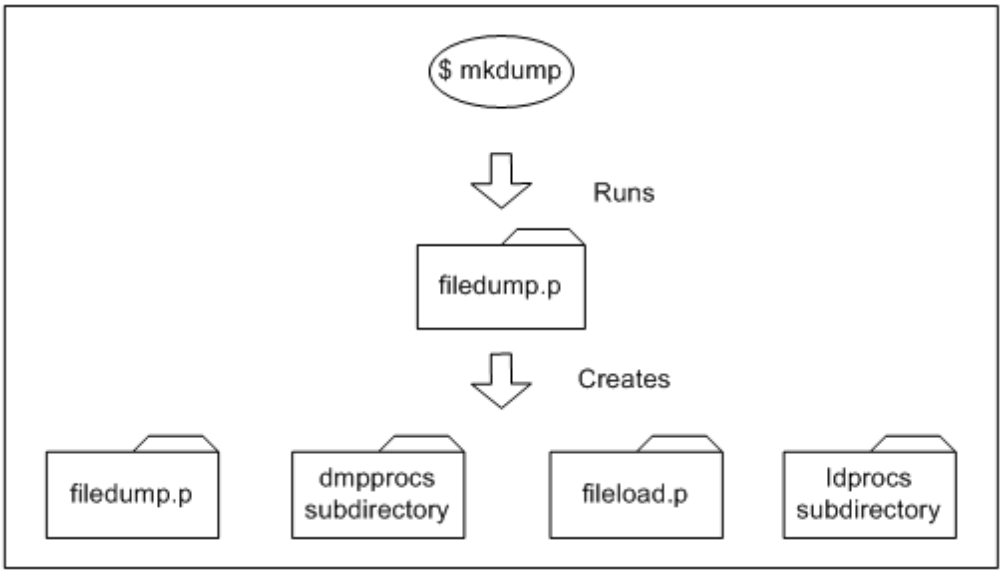
MKDUMP utility

This sample utility contains a dump/load facility. When you run the MKDUMP utility, it creates two ABL procedures: one that dumps a database and one that reloads a database. For example, you can run MKDUMP to create a dump/reload facility for the sports database, as follows:

```
mkdump sports
```

The following image shows the files produced by running MKDUMP.

Figure 13: Output of mkdump



The MKDUMP utility runs an ABL procedure, `mkdump.p`, which creates and compiles two other ABL procedures, `filedump.p` and `fileload.p`. The `filedump.p` procedure dumps the files from a database; `fileload.p` loads the files into a database.

In addition, `mkdump.p` creates two directories. The `dmpprocs` subdirectory contains a database dump procedure for each of the files in the database. The `ldprocs` subdirectory contains a database load procedure for each of the files in the database. The `mkdump.p` procedure also precompiles all the procedures it creates. These precompiled procedures may not be appropriate for your use if the deployed database does not have the same name as the original database that MKDUMP was run against.

The MKDUMP utility creates these files and subdirectories in your current working directory. Before distributing them to users, you must move them to your application master directory. You can then supply either the encrypted source or object versions of the `filedump.p` procedure, `fileload.p` procedure, and each of the procedures in the `dmpprocs` and `ldprocs` subdirectories.

You can modify MKDUMP or any of the `mkdump.p`, `filedump.p`, or `fileload.p` procedures as needed. However, once you have created and perhaps modified the dump/load scripts or batch files, be sure to provide a way for users to use these procedures. You should also provide adequate documentation for users to use the facility.

Note: The dump/load procedures should be considered part of your application. The decision to distribute r-code or encrypted source versions of your application procedures applies to the dump/load procedures as well.

Using your dump/reload facility

To dump and reload a database:

- Use `dumpdb` to dump the current copy of the database.
- Use either `prodb` or `newdb` to make a fresh copy of the basic database.
- Uses `reloaddb` to reload the database.

Before you supply the `filedump.p`,`fileload.p`, and subdirectories of dump and reload procedures to a user, you must understand the following points:

- The `filedump.p` and `fileload.p` procedures were written to run from an interactive ABL procedure. If you plan to have users run these procedures from the operating system command level, you must supply an output destination by adding `OUTPUT TO`*`filename`* at the top of each procedure. Comments in the `dumpdb` and `reloaddb` files explain these modifications.
- If you modify the procedures, remember to recompile.
- If the database to be loaded does not have the same name as the database that the MKDUMP was run against, then you must recompile the `fileload.p` procedure against the database with the proper name.

Restricting database access

OpenEdge provides the DBRSTRCT utility to restrict access to a database. If you restrict the database with DBRSTRCT, you cannot compile procedures or upgrade the database at the user site. You must provide a new empty database and a complete set of `.r` files to upgrade the application, and users must dump and reload their databases.

Because database access restrictions apply to you as well as to your users, you must make a copy of your original, unrestricted and unfrozen database, and store it under a separate name. That way, if you want to lessen the restrictions at a later time, you can dump the data in the restricted database and reload it into a copy of the original, unrestricted database.

You use the DBRSTRCT utility to define the kinds of database access all users, including yourself, are allowed. This is the syntax:

Syntax (UNIX)

```
dbrstrct database-name
```

In these commands, *`database-name`* is the name of the database you want to restrict. The utility starts OpenEdge and displays the following menu from which you can choose options that let you define privileges for the update and query activities:

```
             Database Restriction Utility

This utility lets you make IRREVOCABLE changes to
EVERYONE's ability to modify or query database files.
Before doing this, you should:

         - make a copy of the database
         - precompile all procedures

   Deny (except for precompiled procedures):

1. Update access and/or query access to all current files.
2. Update access and/or query access to individual files.
3. Addition of new files to the database.

                  Selection:
```

To deny update or query access to all database files that are currently defined, choose **1**. A second menu screen appears.

At the second menu, if you choose **1 (Deny updates)**, users can query but not update the database.

Any procedures you compile against a database before restricting that database are exempt from any restrictions you apply with the DBRSTRCT utility.

Make sure that you compile a complete set of dump procedures before you restrict the database. Otherwise, users will not be able to dump their database.

Note: Encrypted source procedures **cannot** be compiled against a restricted database.

A

Building OpenEdge Executables

When you purchase an OpenEdge® product, you receive a set of executable files, database files, and other files that are necessary to run the product on your system. In most cases, the files included in your initial product installation fulfill your system requirements. However, you might need to customize an OpenEdge executable by adding or removing product capabilities. For example, you might need to call a custom C function or access a non-OpenEdge data source. When you need to customize an OpenEdge executable, use the OEBuild utility. The OEBuild utility is a set of scripts that you use to build customized versions of certain OpenEdge executables. This appendix provides information about the process and requirements for building customized OpenEdge executables. It also provides instructions for using the OEBuild utility scripts on UNIX and in Windows.

Note: This appendix describes how to customize an OpenEdge client to access a custom C function. For an OpenEdge client to access an OpenEdge DataServer™ for Oracle®, you must also build a customized executable. For more information, see *Use the Oracle Data Server*.

For details, see the following topics:

- Requirements
- Building executables on UNIX
- Building executables in Windows
- Troubleshooting

Requirements

To use a custom C function in ABL, you must invoke it using the ABL `CALL` statement. This statement accesses your C function in an OpenEdge procedure as part of the Host Language Call (HLC) interface that you run using an OpenEdge executable customized to access this function. This appendix describes how to customize an OpenEdge executable to access a C function using the HLC interface. For more information on using custom C functions in ABL, see the Host Language Call Interface appendix in *OpenEdge External Programming Interface Reference*.

You must ensure that the customers who purchase your product have purchased and installed a properly licensed version of OpenEdge before you install any OpenEdge executables you deliver as part of your product.

Building executables on UNIX

This section provides instructions for building a customized executable on UNIX, as described in the following sections:

- Preparing the UNIX software environment on page 182
- Building a customized executable on UNIX on page 183
- Using the customized UNIX executable on page 184
- Building and running the HLC examples for UNIX on page 185

Preparing the UNIX software environment

When you install OpenEdge, the installation program prepares the software environment. In the UNIX software environment, OpenEdge uses the buildenv script to maintain environment variables.

The OEBuild utility script calls the `buildenv.sh` script. The `buildenv.sh` script sets default values for environment variables that point to objects, libraries, and options involved in linking executables. You can edit this script to customize environments for different sites.

The OEBuild script has the following software and environment requirements:

- Building the executable on the same platform on which it is to be run.
- Installing a linker on your system. The `PATH` environment variable contains the path to the C++ compiler and the linker.
- Setting OpenEdge environment variables. Set `DLC` to the directory where you have installed OpenEdge.
- Including appropriate environment variable settings for your system in the `buildenv.sh` script, which is located in the `$DLC/oebuild/make` directory.

For more information about UNIX environment variables, see *Configure OpenEdge*.

Building a customized executable on UNIX

This section contains instructions for building OpenEdge client or AppServer™ executables running under UNIX to access custom C functions. The OEBuild script is located in the `oebuild/make` directory under the root of your OpenEdge installation.

To build a UNIX OpenEdge client executable:

1. Open a terminal and invoke a Bourne shell (`/bin/sh`) on the system where OpenEdge is installed.
2. Change to user `root`.
3. Run the PROENV utility to set the DLC variable. For example:

```
>$DLC/bin/proenv
```

4. Optionally set and export the environment variable `IMAGE` to the full pathname of the executable to be generated. For example:

```
proenv> IMAGE=/tmp/_myprogres
proenv> export IMAGE
```

Note: By default, the custom client executable is named `$DLC/oebuild/_progres` if you are building the OpenEdge client (or WebSpeed agent) executable. It is `$DLC/oebuild/_proapsv` if you are building the AppServer agent executable.

5. Make sure that the environment variable `PATH` contains the directory to the supported compiler and linker on your system.
6. Unset all the library path variables for your operating system. The script will set these variables as required. For example, on HP-UX:

```
proenv> unset SHLIB_PATH
```

7. Make a backup copy of the original OEBuild link options script and the OpenEdge tie-in object used to access custom C functions, `hlprodsp.o`. There are two versions of the OEBuild link options script, depending on the OpenEdge client executable you plan to build:

   - **`build_rx.sh`** — OpenEdge client (or WebSpeed® agent) executable
   - **`build__proapsv.sh`** — AppServer agent executable

   For example, to copy the original OpenEdge client and OpenEdge tie-in object:

```
proenv> cp $DLC/oebuild/make/build_rx.sh
          $DLC/oebuild/make/build_rx.sh.orig
proenv> cp $DLC/oebuild/obj/hlprodsp.o $DLC/oebuild/obj/hlprodsp.o.orig
```

8. Edit the C source file for the OpenEdge tie-in object in order to tie in your C functions to ABL as explained in the source file comments. This file is located at the following path:

```
$DLC/oebuild/hlc/hlprodsp.c
```

9. Compile the new tie-in object. For example:

```
proenv> cc -c -o $DLC/oebuild/hlc/hlprodsp.o $DLC/oebuild/hlc/hlprodsp.c
```

10. Set and export the environment variable `HLC_OBJS` to contain the paths of all objects that need to be included in your new OpenEdge executable, including the OpenEdge tie-in object you built in Step 9 on page 184. For example:

```
proenv> HLC_OBJS="/home/foo.o /home/foo1.o
$DLC/oebuild/hlc/hlprodsp.o"
proenv> export HLC_OBJS
```

11. Run the OEBuild script located in `$DLC/oebuild/make`, with no arguments. There are two versions of the OEBuild script, depending on the OpenEdge client executable you plan to build:

- **`build_rx.sh`** — OpenEdge character client (or WebSpeed® agent) executable
- **`build_proapsv.sh`** — AppServer agent executable

For example, to build a customized character client use the following command:

```
proenv> $DLC/oebuild/make/build_rx.sh
```

Note: If you do not set the `IMAGE` variable, as in 4 on page 183, the client executable is built as `$DLC/oebuild/_progres` for the OpenEdge character client agent or `$DLC/oebuild/_proapsv` for the AppServer agent; otherwise, the executable is built as specified by `$IMAGE`.

Using the customized UNIX executable

To use the new executable, you must replace the default executable and set appropriate permissions.

To replace your default client executable:

1. Backup the original OpenEdge client executable. For example:

```
proenv> cp $DLC/bin/_progres $DLC/bin/_progres.orig
```

Caution: Do not skip this step. This provides you with a way to rollback your change without reinstalling OpenEdge.

2. Replace the original OpenEdge client executable with a copy of your customized executable. For example, assuming the default setting of `IMAGE`, use the following command:

```
proenv> cp $DLC/oebuild/_progres $DLC/bin/_progres
```

3. Change permissions on the newly created executable to include the Set UID bit.For example:

```
proenv> chmod u+s $DLC/bin/_progres
```

Building and running the HLC examples for UNIX

OpenEdge comes installed with working HLC examples that you can build and test for UNIX. For more information on these examples, see the Host Language Call Interface appendix in *OpenEdge External Programming Interface Reference*.

Building executables in Windows

This section provides instructions for building a customized executable in Windows, as described in the following sections:

- Preparing the Windows software environment on page 185
- Building a customized executable in Windows on page 186

Preparing the Windows software environment

When you install OpenEdge, the installation program prepares the software environment. In the Windows software environment, OpenEdge uses the progress.ini file as well as the Registry to maintain environment variables.

The OEBuild script calls the progress.ini file. The progress.ini file sets default values for environment variables that define objects, libraries, and options required for building executables in Windows. You can edit the progress.ini file to customize environments at different sites.

When you install OpenEdge, the installation program automatically updates the Registry with the information in the progress.ini file that is shipped to you. Progress Software Corporation recommends that you maintain environment variables in both the progress.ini file and the Registry. If you modify environment variables in the progress.ini file, you must update the information in the Registry.

Although there are several methods for adding and deleting information from the Registry directly, Progress Software Corporation recommends that you do not use these direct update methods. Instead, maintain the Registry information by editing the progress.ini file and using the INI2REG utility to update the Registry. This approach synchronizes the information in the progress.ini file and the Registry without causing unintended edits to the Registry.

The OEBuild script has the following software and environment requirements:

- Setting OpenEdge environment variables. Set `DLC` to the directory where OpenEdge is installed.
- Including the appropriate environment variables for your system in the `progress.ini` file (located in the `C:\Progress\OpenEdge\bin` directory, by default) and the Registry.
- Installing the Microsoft® Visual C++® .NET compiler on the system.
- Adding the path to the C++ compiler and the linker command (link.exe) to the `PATH` environment variable.

For information about Windows environment variables, see *Configure OpenEdge*. For more information about maintaining environment variables in the `progress.ini` file and the Registry, and using the INI2REG utility, see the sections on maintaining the Windows user environment in Maintaining User Environments on page 51.

Building a customized executable in Windows

This section contains instructions for building OpenEdge client or AppServer executables running in Windows to access custom C functions. The script is located in the `oebuild/make` directory under the root of your OpenEdge installation.

To build a Windows client executable:

1. Log in as Administrator and run the Proenv utility (**StartProgramsOpenEdge Proenv**) on a system where OpenEdge is installed. This sets the DLC variable.
2. Change directory to the OpenEdge `make` directory. For example:

```
proenv> cd %DLC%\oebuild\make
```

3. Set up the Microsoft Visual C++ environment by running `vcvars32.bat`. For example:

```
proenv> "C:\Program Files\Microsoft Visual Studio        .NET\Vc7\bin\vcvars32.bat"
```

4. Make a backup copy of the original OEBuild link options script and the OpenEdge tie-in object used to access custom C functions, `hlprodsp.o`. There are three versions of the OEBuild link options script, depending on the OpenEdge client executable you plan to build:
   - **`build_prow32.link`** — OpenEdge GUI client executable
   - **`build__prchar.link`** — OpenEdge character client (or WebSpeed® agent) executable
   - **`build__proapsv.link`** — AppServer agent executable

For example, to copy the original GUI OpenEdge client and OpenEdge tie-in object use the following commands:

```
proenv> copy build_prow32.link build_prow32.link.orig
proenv> copy ..\obj\hlprodsp.obj ..\obj\hlprodsp.obj.orig
```

5. Edit the C source file for the OpenEdge tie-in object in order to tie in your C functions to ABL as explained in the source file comments. This file is located at the following path:

```
%DLC%/oebuild/hlc/hlprodsp.c
```

6. Compile the new tie-in object. For example:

```
proenv> cl -c -Fo ..\obj\hlprodsp.obj ..\hlc\hlprodsp.c
```

7. Edit the link options script for your executable (`build_prow32.link`, `build__prchar.link`, or `build__proapsv.link`) to add customized objects and any necessary libraries to accommodate them on the link line. Save the file.
8. Run the `link` command to create a customized OpenEdge client dynamic link library (DLL). For example, to create a customized GUI OpenEdge client DLL:

```
 proenv> link @build_prow32.link
```

9. Back up the original OpenEdge client DLL. For example:

```
proenv> copy %DLC%\bin\prow32.dll %DLC%\bin\prow32.dll.orig
```

10. Copy the customized OpenEdge client DLL into the installed `bin` directory. For example:

```
proenv> copy %DLC%\oebuild\make\prow32.dll %DLC%\bin\prow32.dll
```

Troubleshooting

The following are some common errors that might occur when you attempt to link an executable:

- An object or library file cannot be found.
- An environment variable is not properly set.

- The system has insufficient memory or disk space.

For link utility error and troubleshooting information, see the documentation supplied with your operating system.

B

R-code Features and Functions

R-code is the intermediate binary code that OpenEdge® generates when it compiles ABL (Advanced Business Language) source files. This is the code that OpenEdge actually runs when it executes a procedure or class, whether from session compiles or from permanently generated r-code (.r) files.

Note: Whether the source code is an ABL procedure or ABL class, it compiles into r-code. For the purposes of this topic, procedure and class files are equivalent.

OpenEdge provides a dynamic r-code execution environment, as well as integrity and security mechanisms to ensure that your r-code is running in a compatible environment.

For details, see the following topics:

- R-code structure
- Procedure libraries
- Procedure Libraries and PROPATH
- R-code execution
- R-code portability
- Code page compatibility
- Database CRCs and time stamps
- R-code CRCs and procedure integrity

R-code structure

R-code is divided into multiple segments of varying length. Each r-code file contains an object header and segment location table followed by the actual r-code segments. The object header is a fixed-length descriptor that identifies the file as an r-code file and contains information about the version and size of the r-code file. The segment location table is a variable-length descriptor that contains the size and location of each r-code segment in the file. The maximum size for all segment types is 4MB.

The following table describes and lists the types, maximum size, and maximum number of segments in an r-code file.

Table 30: R-code segments

Segmenttype	Max.size	Max.number	Description
Action code	4MB	1 for main procedure1 per internal procedure	Holds the actual executable code in the form of action cells. Action cells drive the ABL interpreter and contain the executable code for ABL verbs. There is one segment for the main procedure, and one separate segment for each internal procedure.
Expression code	4MB	1	Holds the executable expressions in Reverse Polish Notation (RPN) for the main procedure, all internal procedures, and all trigger blocks."
Text	4MB	1 per language	Holds all literal character strings for the r-code file. There is one text segment for the default language and one for each language specified in the COMPILE statement. Duplicate literals are removed. Only one text segment is loaded into memory at run time.
Initial value	4MB	1	Contains information required to initialize an r-code file for execution, including database and table references, variable definitions, TEMP-TABLE and WORK-TABLE definitions, a list of defined frames, CRCs, and time stamps. There is one initial value segment per r-code file.
Frame	4MB	1 per frame	Contains layout information for a frame, including frame fields, attributes, and all RPN expression code from the frame phrase. There is one frame segment for each named and unnamed frame in the r-code file. Each frame segment has a 32K limit.
Debugger	4MB	1	Used by the Application Debugger to maintain the debugger context for the procedure. There is one debugger segment per r-code file. This segment is loaded into memory on demand only when the Debugger is active. For information about the Debugger, see *Troubleshoot ABL Applications*.

Factors that affect r-code size

Compilation fails if you exceed segment size limits. However, you can affect r-code size by how you write your ABL, depending on the r-code segment. The action code and initial value segments are among the most tunable.

Action code segment

You can reduce the size of an action code segment (for either the main procedure or an internal procedure) by consolidating multiple ABL statements into one. This can also increase the speed of execution, because the interpreter executes only one action instead of several.

For example, you can reduce action code size by combining several consecutive assignment statements into one `ASSIGN` statement. The following examples require the same amount of expression code:

```
a = 1.
b = 2.
c = 3.
```

```
ASSIGN a = 1
       b = 2
       c = 3.
```

Initial value segment

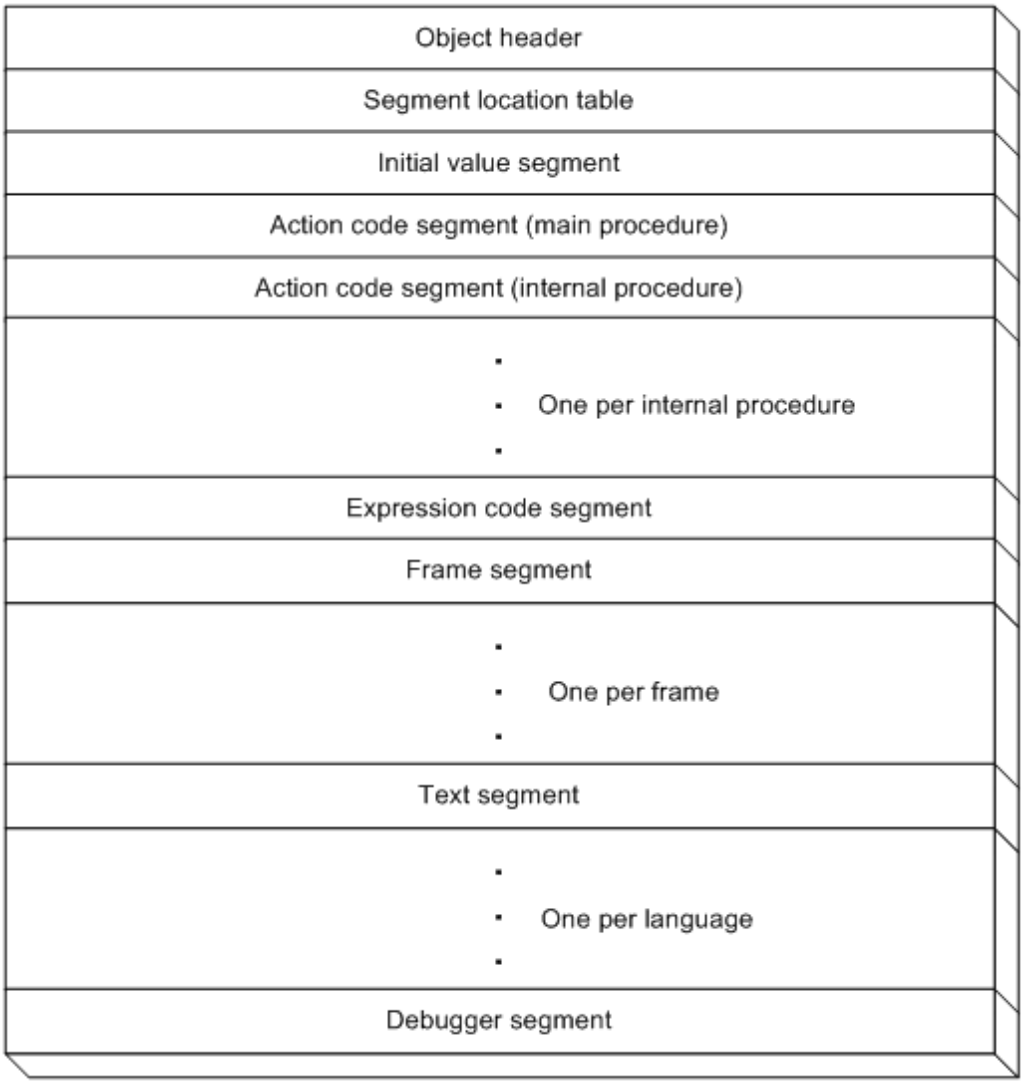
You can reduce the size of the initial value segment by limiting the number of `SHARED` variables that are accessed in a procedure. The r-code required to support a `SHARED` variable is larger than the r-code to support a `NEW SHARED` variable. OpenEdge uses approximately 36 additional bytes in the initial value segment to resolve each `SHARED` variable at run time. This value can change depending on the environment.

R-code file segment layout

The following image shows the segment layout in an r-code file. A compiled procedure requires one initial value segment, one main action code segment, one expression code segment, one text segment, and one debugger segment. There can be more action and expression code segments, up to the limit, as required by your procedure. There can be multiple frame segments, one segment for each frame in your procedure. The maximum number of frame segments is virtually unlimited.

There can also be multiple text segments, one segment for the default language and each language that you choose for translation. The default language segment contains the literal character strings defined in the original ABL source file. You can create additional text segments by using the Translation Manager. Although multiple text segments are possible, only one text segment is available per language. Thus, the literal character strings for a given language cannot exceed 4MB.

Figure 14: R-code file segment layout



Procedure libraries

You can organize and store r-code files in an ABL procedure library. An ABL procedure library is a collection of r-code and images combined in a single file. Procedure libraries allow you to manage and execute r-code more efficiently. You create procedure libraries by using the PROLIB utility.

Note: OpenEdge can only make use of r-code and images within a procedure library. It is technically possible to include other types of files in a procedure library. However, the user would need to extract them from the library before making use of them.

OpenEdge provides two types of procedure libraries: standard and memory-mapped. A standard library contains r-code that execute in local memory. A memory-mapped library contains r-code that execute in shared memory.

For information about using the PROLIB utility to create procedure libraries, see Managing Procedure Libraries on page 115.

Procedure Libraries and PROPATH

The following rules govern how standard and memory-mapped libraries interact with PROPATH during an OpenEdge session:

- The first time you run a member from a standard or memory-mapped library that is specified in the PROPATH, OpenEdge locates the member by searching through each directory and library in the PROPATH until it finds the member. To search a library for a member, OpenEdge must open the library. When OpenEdge opens a standard library, it loads the member into local memory. When OpenEdge opens a memory-mapped library, it maps the library in shared memory.
- When searching through directories and libraries in the PROPATH, OpenEdge starts at the beginning of the PROPATH and searches each directory and library, in defined order, until it finds the member. Thus, placing the library at the beginning of the PROPATH improves performance.
- A library remains open until the end of your OpenEdge session or until you remove the library from the PROPATH. If you remove a library from the PROPATH while a member is still active (either running, or waiting for a subprocedure to return), OpenEdge displays an error message and the library remains open until you end your OpenEdge session. Otherwise, OpenEdge closes a standard library or unmaps a memory-mapped library.
- If you use a `SEARCH` or `RUN` statement to open a library that is not in the PROPATH, the library remains open until you end your OpenEdge session.
- If you use libraries, OpenEdge accesses r-code as if you had specified the Quick Request (`-q`) startup parameter. That is, OpenEdge searches the PROPATH only on the first reference to a member. Thereafter, if the member still resides in memory, in the local session compile file, or in an r-code library, OpenEdge reuses the member instead of searching the PROPATH again. For more information about the Quick Request (`-q`) startup parameter, see *Startup Command and Parameter Reference*.

R-code execution

OpenEdge executes r-code in different ways, depending on whether you store the r-code files in an procedure library and the type of library.

When executing r-code from an operating system file in a directory, or from a standard procedure library, OpenEdge accesses and executes the r-code file segments in an execution buffer in local memory. For more information about the standard r-code execution environment, see Standard r-code execution environment on page 194.

When executing r-code from a memory-mapped procedure library, OpenEdge accesses and executes the r-code segments in shared memory. For more information about the memory-mapped r-code execution environment, see Memory-mapped r-code execution environment on page 196.

For information about monitoring execution environment activity during an OpenEdge client session, see R-code execution environment statistics on page 198.

For information about monitoring and optimizing r-code performance, see Managing Client Performance on page 105.

Standard r-code execution environment

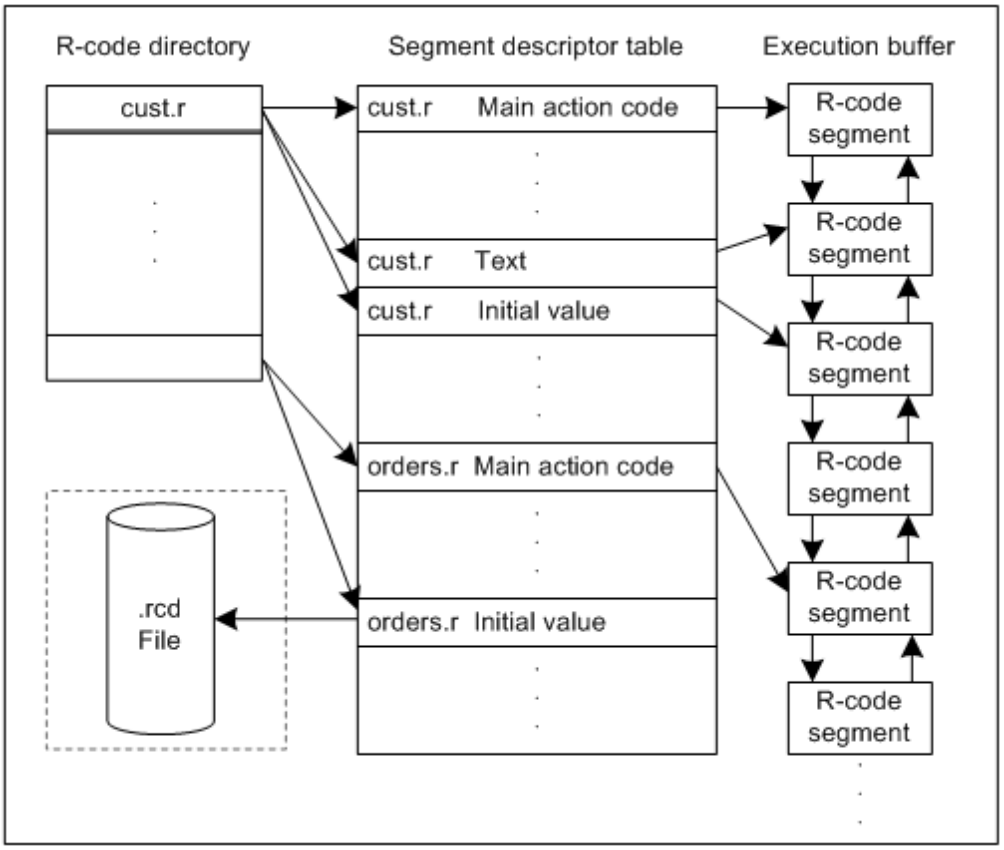
At run time, OpenEdge manages the execution of r-code from either an operating system file or a standard procedure library using the following components:

- **Execution buffer** — The portion of local memory that OpenEdge allocates and uses to store the chain of loaded r-code segments for all procedures executing in local memory.
- **R-code swap file (.rcd)** — A file that OpenEdge uses to dynamically swap r-code segments in and out of the execution buffer.
- **Segment descriptor table** — An in-memory table that references the r-code segments required by all executing procedures, including the location of each r-code segment in the execution buffer and usage count information.
- **R-code directory** — An in-memory table that contains information about each executing r-code, including r-code size, usage count, segment descriptions, and a reference to the segment descriptor table for each segment.

OpenEdge uses the segment descriptor table and the r-code directory to manage r-code from operating system files, standard libraries, and memory-mapped libraries.

The following image shows the layout for the standard r-code execution environment.

Figure 15: Standard r-code execution environment



In the above image, OpenEdge located and loaded the `cust.r` and `orders.r` files into local memory from either operating system files or standard libraries. The execution buffer is shown with three `cust.r` segments and two `orders.r` segments. Note that one `orders.r` segment is located in the execution buffer, while the other segment is swapped to the r-code swap file (`.rcd`). When space in the execution buffer is needed for new r-code segments, OpenEdge uses the r-code swap file to swap out the least-recently used segments. When OpenEdge needs a segment that has been swapped to the r-code swap file, it reloads the segment from the r-code swap file into the execution buffer.

Standard execution sequence

When you run r-code for the first time, from either an operating system file or a standard library, OpenEdge loads and executes the r-code as follows:

1. Opens the file (`.cls`, `.p`, or `.r`) or procedure library (`.pl`), if the library is not already open.
2. Reads the r-code into memory and creates an r-code directory entry for it. OpenEdge first compiles a procedure into r-code, if necessary.
3. Registers each required r-code segment in the execution environment as follows:
   a) Loads the r-code segment at the head of the segment chain in the execution buffer
   b) Adds an r-code segment entry to the segment descriptor table that references the segment in the execution buffer
   c) Inserts a segment descriptor reference in the r-code directory entry for the r-code

   If all the required r-code segments do not fit in the execution buffer, OpenEdge attempts to free space by swapping r-code segments already in the buffer to the r-code swap file. If OpenEdge cannot free enough space by swapping segments, it increases the execution buffer ceiling and allocates more space for the execution buffer.

   Note: Codepage-converted text segments within standard or memory-mapped procedure libraries are swapped in and out of the r-code swap file, preserving any code page conversions that occurred during the initial load of the text segment. When accessing members stored in a standard library, OpenEdge does not swap r-code segments to the r-code swap file unless you specify the PROLIB Swap (`-pls`) startup parameter. By default, if OpenEdge needs an r-code segment in a standard library, it reloads the segment into the execution buffer from the library in local memory.

4. Once the required r-code segments are registered in the execution environment, the interpreter begins executing the r-code at the start of the first main action code segment and accesses the remaining segments directly from local memory as required.

Standard execution environment limits

The amount of r-code that you can run at one time and how much memory it uses are determined by the following factors:

- **R-code directory size** — The default is 100 entries, the minimum is five entries, and the maximum is 2,147,483,647. You can set the initial number of entries using the Directory Size (`-D`) startup parameter. OpenEdge dynamically increases the directory size up to the maximum, as required. Use the Hardlimit (`-hardlimit`) startup parameter to force OpenEdge to adhere to the limit specified by the Directory Size (`-D`) startup parameter.
- **Execution buffer ceiling** — The default is 3096K. You can set the initial ceiling for this buffer up to 65,534K using the Maximum Memory (`-mmax`) startup parameter. OpenEdge dynamically increases the execution

buffer size up to the maximum, as required. Use the Hardlimit (`-hardlimit`) startup parameter to force OpenEdge to adhere to the limit specified by the Maximum Memory (`-mmax`) startup parameter.

- **Available memory** — Available memory is a factor only if it is smaller than the execution buffer ceiling or OpenEdge needs to allocate memory beyond that ceiling.

Standard r-code segment management

While OpenEdge loads r-code, all of its segments are locked in memory. After all required segments are loaded from the r-code, OpenEdge unlocks all segments except its main action code segments and text segment. These segments stay locked in memory until execution of the r-code terminates. Internal procedure action code segments stay locked only until they return to the invoking procedure and are relocked each time they execute.

When a standard r-code segment does not fit in the execution buffer, OpenEdge attempts to free space by swapping r-code segments already in the buffer to the r-code swap file. OpenEdge can swap out any unlocked segments and then remove these segments from the tail end of the execution buffer chain, in least recently used (LRU) order.

If OpenEdge cannot free enough memory for a newly loaded segment by swapping out older segments, it dynamically increases the maximum execution buffer size (execution buffer ceiling) and allocates the required memory up to the memory available.

When OpenEdge needs a segment that has been swapped to the r-code swap file, it reloads the segment from the swap file into the execution buffer. However, OpenEdge keeps the reloaded segments in the swap file.

Note: The AVM swaps codepage-converted text segments for procedure libraries from the execution buffer to the r-code swap file and reloads the converted text segments from the swap file instead of the library when necessary. When accessing members stored in a standard library, OpenEdge does not swap r-code segments to the r-code swap file unless you specify the PROLIB Swap (`-pls`) startup parameter. By default, if OpenEdge needs an r-code segment in a standard library, it reloads the segment into the execution buffer from the library in local memory.

Memory-mapped r-code execution environment

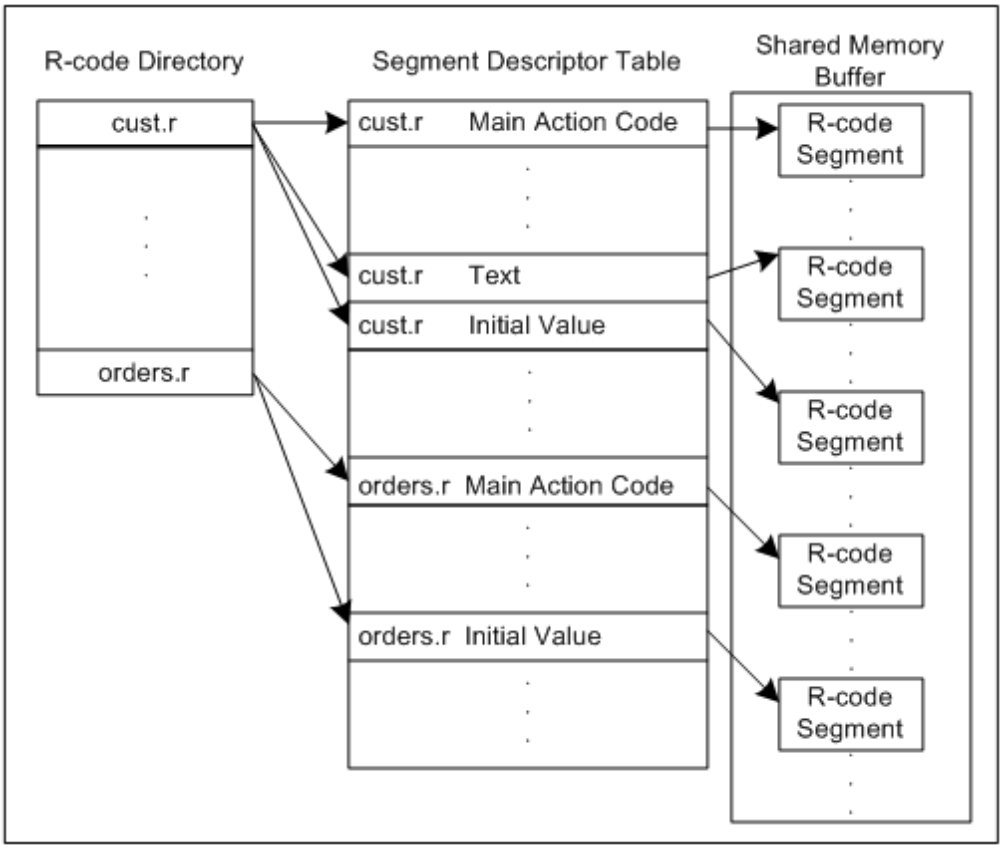
At run time, OpenEdge manages the execution of memory-mapped r-code using the following components:

- **Shared memory buffer** — The portion of shared memory that the operating system allocates and uses to store r-code segments for members in a memory-mapped procedure library
- **Segment descriptor table** — An in-memory table that references the r-code segments required by all executing members, including the location of each r-code segment in memory and usage count information
- **R-code directory** — An in-memory table that contains information about each executing member, including r-code size, usage count, segment descriptions, and a reference to the segment descriptor table for each segment

OpenEdge uses the segment descriptor table and the r-code directory to manage r-code from operating system files, standard libraries, and memory-mapped libraries.

The following image shows the layout for the memory-mapped r-code execution environment.

Figure 16: Memory-mapped r-code execution environment



In the above image, OpenEdge locates the `cust.r` and `orders.r` members in a memory-mapped library. Note that all r-code segments in a memory-mapped library are located in shared memory. When OpenEdge needs a segment, it executes the segment directly from shared memory.

Memory-mapped execution sequence

When you run a memory-mapped member for the first time, OpenEdge loads and executes the r-code as follows:

1. Opens the procedure library, if not already open, and memory-maps the library in shared memory.
2. Reads the member and creates an r-code directory entry for the member.
3. Registers each required r-code segment in the execution environment as follows:
   a) Adds an r-code segment entry to the segment descriptor table that references the segment in shared memory
   b) Inserts a segment descriptor reference in the r-code directory entry for the member
4. Once the required r-code segments are registered in the execution environment, the interpreter begins executing the r-code at the start of the first main action code segment and accesses the remaining segments directly from shared memory as required.

Memory-mapped execution environment limits

The amount of memory-mapped r-code that you can run at one time and how much shared memory it uses are determined by the following factors:

- **R-code directory size** — The default is 100 entries, the minimum is five entries, and the maximum is 2,147,483,647. You can set the initial number of entries using the Directory Size (`-D`) startup parameter. OpenEdge dynamically increases the directory size up to the maximum, as required. Use the Hardlimit (`-hardlimit`) startup parameter to force OpenEdge to adhere to the limit specified by the Directory Size (`-D`) startup parameter.
- **Available memory** — The number of memory-mapped procedure libraries you can have open is limited by the amount of shared memory available on the system.

Memory-mapped r-code segment management

When OpenEdge needs a memory-mapped r-code segment, it executes the segment directly from shared memory. OpenEdge does not store active memory-mapped r-code segments in the execution buffer. Nor does it swap non-active segments to the r-code swap file. OpenEdge relies on the operating system to manage the swapping of r-code segments in and out of shared memory.

Note: The AVM swaps codepage-converted text segments for procedure libraries from the execution buffer to the r-code swap file and reloads the converted text segments from the swap file instead of the library when necessary.

R-code directory management

If the r-code directory is full when loading additional r-code from any source, OpenEdge can reuse existing directory entries for any r-code files no longer in use.

To reuse an r-code directory entry, OpenEdge performs the following steps:

1. Identifies the least recently used (LRU) r-code directory entry.
2. Frees the LRU r-code directory entry.
3. Frees all of the segment descriptor entries in the segment descriptor table for the LRU r-code directory entry.
4. When reusing a standard r-code directory entry, OpenEdge removes all r-code segments for the LRU r-code file from both the execution buffer and the r-code swap file. When reusing a memory-mapped r-code directory entry, the memory-mapped r-code file remains mapped in shared memory.

   When OpenEdge needs to reload standard r-code, it follows the standard execution sequence. When OpenEdge needs to reload memory-mapped r-code, it follows the memory-mapped execution sequence.

R-code execution environment statistics

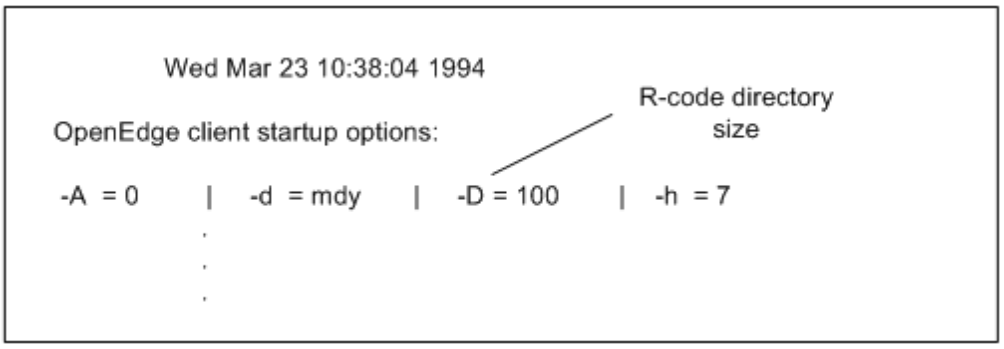
You can monitor execution environment activity using the Statistics (`-y`) and Segment Statistics (`-yd`) startup parameters. These parameters cause OpenEdge to write memory statistics to the `client.mon` file. The `-yd` parameter provides all of the information about memory usage available with `-y` plus additional information about r-code segments loaded during a client session. For more information about the `-yd` and `-y` startup parameters, see *Startup Command and Parameter Reference*.

For information about monitoring and optimizing r-code performance, see Managing Client Performance on page 105.

The following images show sections of `client.mon` output generated with the `-yd` startup parameter.

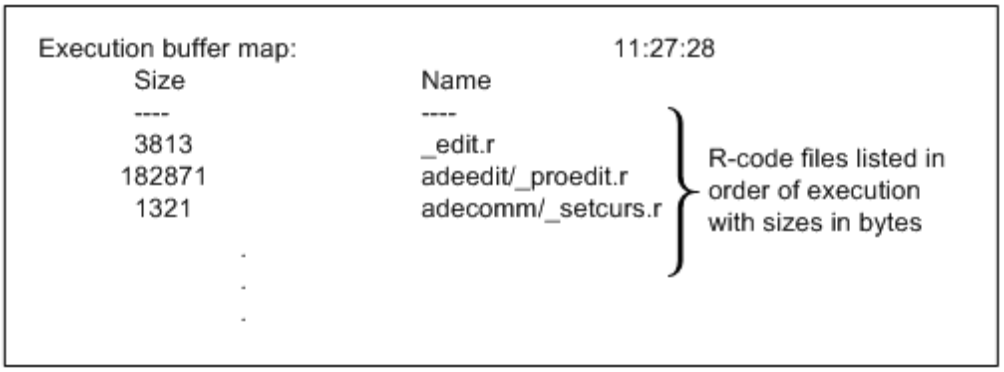
In the following image, the "OpenEdge client startup options" section shows the r-code directory size (`-D`), in this case, set to the default of 100 entries. The `-y` startup parameter also generates this information.

Figure 17: OpenEdge client startup options for –yd



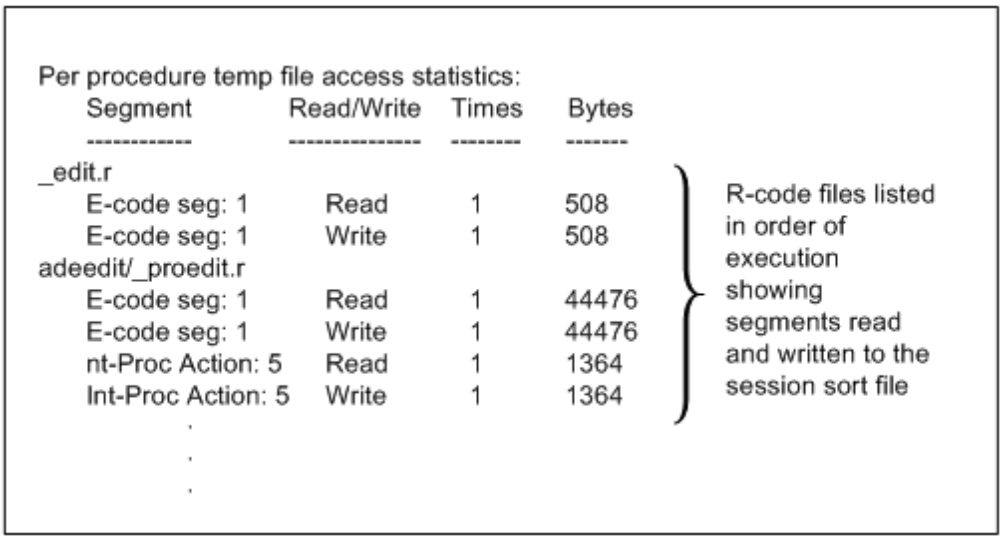
In the following image, the "Execution buffer map" section shows the order and size of procedures loaded into the execution buffer.

Figure 18: Execution buffer map for –yd



In the following image, the "Per procedure temp file access statistics" section lists each r-code segment that has been read or written to the r-code swap file (`.rcd`). Each segment is listed under its r-code file by segment type ("Int-Proc Action"), number (5), and size (1364 bytes).

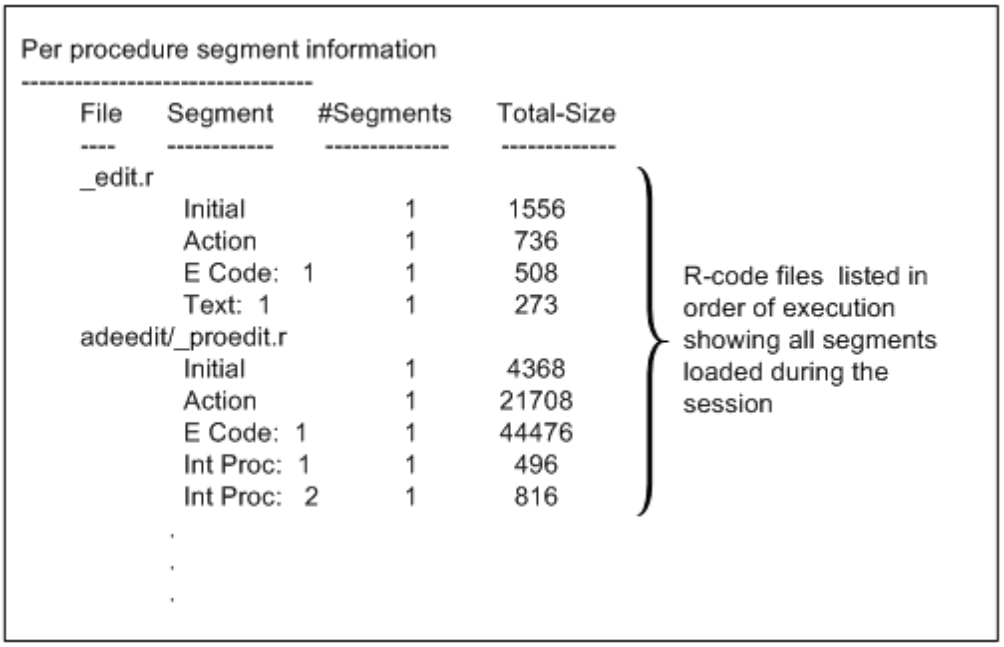
Figure 19: Accessing the r-code swap file for –yd



In this example, all segments shown have been read and written once. A large number of segments read or written a large number of times indicates a likely need for more memory.

In the following image, the "Per procedure segment information" section lists all the r-code segments loaded during the session. Each segment is listed under its r-code file by segment type ("Int-Proc"), number (2), and size (816 bytes).

Figure 20: Procedure segment information for –yd (Part 1)



Thus, if there are three internal procedure action segments in `appedit/_proedit.r`, they are listed in order (Int-Proc: 1 through 3). Likewise, multiple main action code segments are listed in order (A-Code: 1 and 2). The number of segments for each entry is always one. Note that the "Initial" segment has no segment number, because there is never more than one of this segment type per r-code file.

In the following image, the listed sections provide memory and segment usage summaries. The `-y` startup parameter also generates this information.

Table 31: Procedure segment information for –yd (Part 2)

```
------------------------------------------
Program access statistics:    Times    Bytes
                     .
                     .
                     .
Memory usage summary:       Current    Max Used       Limit (Bytes)
Stack usage (-s):                60        7248       40960
Local buffer usage:             864       14704
R-code Execution Buffer:     530025      710609      710656
Segment Descriptors Usage: (numbers)
Max Used:      508      Limit:      720
------------------------------------------------
```

The "R-code Execution Buffer" statistics show how far your application pushes the execution buffer ceiling. The "Segment Descriptors Usage" statistics shows how close your application is to running out of segment descriptors, and thus, whether you need to optimize the number of r-code segments in your application.

R-code portability

You can often port the r-code without recompiling. However, you must recompile if one of the following changes:

- The display architecture
- The database type
- The r-code version

Display architectures

The two display architectures supported by OpenEdge are graphical and character. For information about deploying web applications, see *Develop Web Clients*.

Code that does not contain user-interface statements (specifically, code that does not create frames, either explicitly or implicitly) does not belong to any display architecture, and so does not involve a change of display architectures. If a procedure contains no user interface code, its r-code can run across user interfaces without change. This is typical, for example, of database schema trigger procedures. Another example of such code is a batch program.

Database types

Database types are OpenEdge, ORACLE, and MS-SQL server. Applications access non-OpenEdge databases through the appropriate OpenEdge DataServer products.

Code that does not access any OpenEdge or non-OpenEdge database does not belong to any database type, and so does not involve a change of database types. An example of such code is a date-conversion procedure.

R-code version

Porting across r-code versions sometimes requires recompiling. The r-code version always changes between major releases. It might or might not change between minor point releases depending on the added functionality.

If you port to:

- A lower or higher major release, you must recompile.
- A higher point release within a major release, you might or might not need to recompile, depending on the new functionality in the point release.
- A lower point release within a major release, you might or might not need to recompile. However, it is highly recommended that you recompile.

R-code portability and operating system-specific statements

If you want your r-code to be portable, avoid OpenEdge statements that are operating-system specific. For example, the `DDE INITIATE` statement is specific to Windows.

Byte order of integers in R-code

R-code, whether on disk or in memory, always stores integers in byte-swapped order, even on machines that store integers in byte-stream format. On such machines, OpenEdge converts integers to byte-stream format at run time.

Examples of portable and nonportable r-code

Not all r-code is portable. You can usually port r-code between clients as long as you do not change display architectures, database types, platform classes or major r-code versions. For example:

- If you port a Windows application to character-mode, you must recompile, because the display architecture has changed.
- If you port a character application compiled against an ORACLE DataServer on a SUN workstation to a character application accessing an ORACLE DataServer on a Hewlett-Packard (HP) workstation, you do not have to recompile (assuming that the ORACLE release level does not change and both platforms are 32-bit or 64-bit, respectively).
- If you port a character application compiled against the DataServer for ORACLE on a SUN workstation to a character application accessing the DataServer for Microsoft SQL Server on an HP workstation, you must recompile because the database type has changed.
- If you migrate an application from either Progress® Version 8 or Progress Version 9 to OpenEdge Release 10, you must recompile it. If you want to migrate a Progress Version 8 application to OpenEdge 10, it is not necessary to upgrade to Version 9 first, you can upgrade from Progress Version 8 directly to OpenEdge Release 10.

You must be able to reproduce the target environment to generate the required r-code. For each combination, you have to keep track of a different code tree. For each combination, you also require a full development OpenEdge product to do the compilations.

Code page compatibility

The Internal Code Page (`-cpinternal`) startup parameter determines the internal code page that OpenEdge uses in memory. The code page in which r-code is compiled and the internal code page of the session in which the r-code runs must be compatible. That is, the r-code code page must be either the internal code page, undefined, or convertible to the internal code page.

If the code page for r-code, from either an operating system file, a memory-mapped procedure library, or a standard procedure library, is not the same as the session's internal code page, OpenEdge performs an in-memory conversion of the r-code text segments to the session's internal code page.

Note: If you use the R-code In Code Page (`-cprcodein`) startup parameter to specify a code page for reading r-code text segments, OpenEdge reads the text segments as if they were written in that code page, even if the text segments were written in a different code page.

When executing r-code from a standard procedure library, the ABL Virtual Machine (AVM) accesses and executes the r-code segments in the execution buffer in local memory. The AVM does not swap the r-code segments to the r-code swap file unless you specify the PROLIB Swap (-pls) startup parameter. By default, the AVM reloads segments from the open library in local memory.

Codepage-converted text segments from memory-mapped and standard procedure libraries are swapped in and out of the r-code swap file, preserving any code page conversions that occurred during the initial load of the text segment.

For more information about the Internal Code Page (`-cpinternal`) and R-code In Code Page (`-cprcodein`) startup parameters, see *Startup Command and Parameter Reference*.

Database CRCs and time stamps

At run time, OpenEdge must ensure that r-code files and the database(s) that they access are compatible. Otherwise, if the schema has changed since the code was compiled, running the code might damage the database. OpenEdge supports the cyclic redundancy check (CRC) validation technique for checking this compatibility. This technique prevents you from running unauthorized or data-incompatible procedures against a particular database environment.

CRC validation

For each table, OpenEdge computes a CRC value (*`database`* CRC) from selected elements of the table metaschema. When compiling a procedure, OpenEdge inserts the database CRC for each database table the procedure accesses. The database CRC for each table is stored in the metaschema `_File` record `_CRC` field for the table. A matching CRC ensures that the schema referenced by the r-code is compatible with the table schema.

CRCs help maintain database integrity. CRCs change only when certain schema elements critical to field or record definitions change. As long as a table is structurally compatible with the r-code that references it, the CRCs match and the r-code can execute. It does not matter when or where the table's schema was created or updated.

CRC advantage

The main advantage of using CRC validation is that you can run the same r-code against different databases as long as the databases have sufficiently similar schemas. This allows you to deploy code and database updates while avoiding either of the following effects:

- Forcing the user to dump and reload their data into a new database that you supply with new r-code.
- Forcing the user to compile new source code (possibly encrypted) that you supply with new data definitions to update their database schema.

Depending on the schema changes, the user still might have to dump and reload some data, but you can always supply r-code for their updated database.

R-code execution with CRCs

When executing a procedure using CRC validation, OpenEdge follows these steps:

1. For each table accessed by the r-code, check the time stamp. If it matches, execute the procedure. Otherwise, go to Step 2.
2. Check the CRC. If it matches, execute the procedure. Otherwise, reject the procedure.

   Thus, if the time stamps match, OpenEdge assumes that the database schema and r-code must be compatible. Otherwise, the CRCs determine the outcome.

CRC calculation

The following table lists the metaschema fields that are involved in each CRC calculation.

Table 32: Metaschema fields in CRC calculation

Metaschema table			
_File	**_Field**	**_Index**	**_Index–Field**
_File-Name _DB-lang	_Field-Name _Data-Type _dtype _sys-field _field-rpos _Decimals _Order _Extent _Fld-stdtype _Fld-stlen[2] _Fld-stoff _Fld-case	_Index-Name _Unique _num-comp	(_Field-Name)[3] _Ascending _Abbreviate _Unsorted

CRC calculation for tables

All of the listed metaschema fields participate in the CRC calculation for each database table. The resulting CRC is stored in the _CRC field of the _File record for each table. A change to any of these fields automatically changes the value of _CRC.

Database changes that affect CRCs

You must recompile procedures that reference a database table in which you have made any of the following changes:

- Deleted the table
- Changed the table name
- Changed the database language or authorization key
- Added or deleted a field in the table
- Changed the name or order of a field

Note:

Changes in order or position of the fields in tables of the dataserver database do not require recompilation of R-code.

The order in which indexes are defined has no effect on database CRCs.

In addition to the schema changes listed above, you must recompile procedures that reference a database table in which you have made the following changes to a field:

- The data type or extent (array size)
- The number of decimals

[3] The _Field–Name for index fields is referenced in the _Field table using _Field–recid in _Index–Field.
[2] The _Fld-stlen is not involved in the CRC calculation for BLOB and CLOB fields.

- The case sensitivity

Although you do not have to recompile procedures when you make other schema changes, such changes do not apply to procedures that you do not recompile. For example, changes to the following features appear only when you recompile:

- **Field appearance** — Format, label, column label, and decimal
- **Data entry** — Initial value, mandatory value, case sensitivity, validation expression, delete validation, and security
- **User information** — Help messages, validation messages, and description

Database CRC changes that do not prevent r-code execution

Under certain circumstances, database modifications can change the database CRC value but will not require you to recompile existing r-code. You can run r-code without recompiling if:

- You add new field to a table with a `_Field-rpos` value greater than any other field in the table. In other words, the new field cannot be inserted into an existing `_Field-rpos` sequence.
- You make changes to existing fields that **do not** affect the values of `_Field-name`, `_dtype`, `_Field-rpos`, `_Extent`.

Note: The above exceptions do not apply to DataServer tables. However, changes in field positions in tables of the DataServer database do not require recompilation of r-code.

CRC and time stamp availability

During compilation, OpenEdge always inserts both database time stamps and CRCs in the r-code. OpenEdge marks the r-code to check the CRC if the time stamp check fails.

CRC-based deployment

Using CRC validation, if you deploy r-code, you only have to compile the procedures against a database with the same basic structure as the end-user database. It does not matter when or where the database was created. If you make schema changes to an existing database, you can distribute the newly compiled r-code to end users, along with a data definition files to install the schema changes.

The flexibility of CRC-based deployment makes it easier to distribute a new version of an existing application.

To deploy an application update based on CRC validation:

1. Install a copy of the production database from the target site in your development environment.
2. Create an incremental data definition file that captures the difference between your development database and the production copy. For information about creating an incremental data definition file, see *Manage the OpenEdge Database*.
3. Apply the incremental data definition file to your copy of the production database.
4. Recompile all procedures affected by the schema changes against your copy of the production database.
5. Apply the incremental data definition file to the original production database (on site). The CRC for the original production database should now be identical to the production copy, but probably not identical to your development database.
6. Distribute the new r-code files (from Step 4) to the updated production site.

The steps for CRC-based deployment are essentially the same as those for time stamp-based deployment, but you gain these advantages:

- You are not required to encrypt and give end users source code.
- End users do not have to compile your source code.
- You do not have to send a copy of a new database to end users.
- End users do not have to dump and reload all their data into your new database.

R-code access authorization

One possible drawback of using CRC validation is that anyone having an OpenEdge development environment can create a counterfeit database (a database with the same structure as your database). The user can then write a program, compile it, and run it against your database.

However, you can combine CRC validation with additional security to prevent unauthorized r-code access. You can either turn on run-time permission checking or use the PROUTIL utility's `DBAUTHKEY` qualifier. For details on how to turn on run-time permission checking, see Maintaining Application Security on page 39.

The `DBAUTHKEY` option of the PROUTIL utility allows you to set an authorization key for your database. When you compile your source code, OpenEdge includes the value of this authorization key in your r-code. You can also use the `RCODEKEY` option of PROUTIL to insert the authorization key in existing r-code. Any r-code that does not include the correct authorization key cannot run against your database. For more information about using PROUTIL to create and set database authorization keys, see *OpenEdge Data Management: Database Administration*.

R-code CRCs and procedure integrity

When OpenEdge compiles a procedure, it calculates a special CRC based on the procedure name and code content, and stores it in the r-code. This *r-code CRC* (as distinguished from database CRC) is useful to ensure the integrity and security of ABL procedures, especially for schema triggers. In addition, you can use the r-code CRC to determine whether you must recompile a procedure file after a database change or between different versions of your application.

If you choose to have CRCs checked for a schema trigger, the r-code CRC of the trigger procedure definition is stored in the trigger schema. The trigger procedure cannot run unless it has a matching r-code CRC. This prevents a trigger procedure with the same name, but different ABL code, from being improperly substituted for the original trigger procedure. Such a substitution can cause damage to your database or override your security.

For other application r-code files, you can use the `RCODE-INFO` handle to build a procedure security table that contains the name and r-code CRC of each r-code file in your application (except the startup procedure). Before running a procedure, your application can use the `RCODE-INFO` handle to validate the r-code CRC of its r-code file against the entry established for it in the procedure security table.

Note: If there is no r-code for the procedure, any source (`.p`) procedure returns an Unknown (?) CRC value.

Assigning CRCs to schema triggers

When you create a schema trigger in the Data Dictionary, you can specify CRC checking. Then, when you save the trigger procedure, the Data Dictionary calculates its r-code CRC and stores it in the trigger metaschema record (`_File-Trig` for table triggers and `_Field-Trig` for field triggers).

If you want CRC checking for all your schema triggers, you must specify it for each schema trigger in your database. For more information, see *OpenEdge Development: Basic Development Tools* (Character only) and, in graphical interfaces, the on-line help for the Data Dictionary.

Validating CRCs for schema triggers

When a database event occurs in your application (such as creating a record or assigning a field), OpenEdge checks whether the CRC of the corresponding trigger procedure matches the CRC in the trigger metaschema record. If it matches, OpenEdge executes the procedure. Otherwise, OpenEdge returns a run-time error.

Trigger CRC validation works for both r-code and source versions of a trigger procedure. For source versions, OpenEdge calculates the CRC during the session compile.

RCODE-INFO handle

The `RCODE-INFO` system handle lets you read the r-code CRC from an r-code file. You can use this value to:

- Build a procedure security table, and then to verify the integrity of your application r-code against it
- Automatically install database schema trigger definitions in a deployment procedure that you run in a secure context
- Determine if you need to recompile procedure files after a database change
- Determine if procedure files have changed between different versions of your application

Getting the r-code CRC

To read the r-code CRC from an r-code file, first set the `FILE-NAME` attribute of the `RCODE-INFO` handle to the pathname of your r-code file. Then read the value of the `CRC-VALUE` attribute, as shown:

```
DEFINE VARIABLE rCRC AS INTEGER NO-UNDO.
ASSIGN
  RCODE-INFO:FILE-NAME = "sports/crcust.r"
  rCRC                 = RCODE-INFO:CRC-VALUE.
```

Note: The `RCODE-INFO` handle cannot read the r-code CRC from a session compile. For source (`.p`) procedures, the `CRC-VALUE` attribute returns the `Unknown value (?)`.

Example: Verifying application r-code integrity

Be sure to verify the integrity of your application's r-code.

To provide r-code integrity and security in your application:

1. Build the procedure security table, RcodeSecurity, with these fields:
   - **Filename** — `CHARACTER` field for the pathname of each r-code file
   - **CRC** — `INTEGER` field for the r-code CRC of the specified r-code file
2. Construct a text file, `crctable.dat`, that contains a list of the pathnames, relative to the working directory, for all the secure procedures called by your application.
3. Compile and save the r-code for all the secure procedures in your application.
4. Run a procedure that contains code to build the RcodeSecurity table. For example:

```
DEFINE VARIABLE cProcName AS CHARACTER NO-UNDO FORMAT "x(32)".
DEFINE VARIABLE ix        AS INTEGER   NO-UNDO.
INPUT FROM "crctable.dat". /* List of r-code file pathnames */
REPEAT:
  SET cProcName.
  FIND RcodeSecurity WHERE RcodeSecurity.Filename = cProcName NO-ERROR.
  IF NOT AVAILABLE(RcodeSecurity) THEN
    CREATE RcodeSecurity.
  ASSIGN
    RCODE-INFO:FILE-NAME   = cProcName.
    RcodeSecurity.Filename = cProcName
    RcodeSecurity.Crc      = RCODE-INFO:CRC-VALUE
    ix                     = ix + 1.
END.
INPUT CLOSE.
MESSAGE ix "procedure security records created".
```

5. At each point where you call a secure procedure in your application, insert this code:

```
FIND RcodeSecurity WHERE RcodeSecurity.Filename = "secret.r".
RCODE-INFO:FILE-NAME = "secret.r".
IF RcodeSecurity.Crc = RCODE-INFO:CRC-VALUE THEN
  RUN secret.
ELSE DO:
  MESSAGE "Procedure secret.r is invalid.".
  QUIT.
END.
```

Example: Deploying schema triggers

The `trload.p` procedure installs new schema trigger definitions in an existing database. It reads dump files that contain the new data for table and field metaschema trigger records and updates these records with the new trigger procedure names and CRCs.

trload.p

```
/* trload.p */
DEFINE VARIABLE event      AS CHARACTER NO-UNDO FORMAT "x(6)".
DEFINE VARIABLE field-name AS CHARACTER NO-UNDO FORMAT "x(32)".
DEFINE VARIABLE iCRC       AS INTEGER   NO-UNDO.
DEFINE VARIABLE ix         AS INTEGER   NO-UNDO.
DEFINE VARIABLE proc-name  AS CHARACTER NO-UNDO FORMAT "x(32)".
DEFINE VARIABLE table-name AS CHARACTER NO-UNDO FORMAT "x(32)".
```

```
INPUT FROM "_file-tr.dat". /* Table trigger data */
_fl-loop:
REPEAT:
  SET table-name event proc-name.
  ASSIGN
    RCODE-INFO:FILE-NAME = proc-name
    iCRC                 = RCODE-INFO:CRC-VALUE.
  FIND _file WHERE _file._file-name = table-name.
  FIND _file-trig WHERE _file-trig._file-recid = RECID(_file)
    AND _file-trig._event = event NO-ERROR.
  IF AVAILABLE _file-trig THEN DO:
    IF  _file-trig._proc-name = proc-name
      AND _file-trig._trig-crc  = iCRC THEN NEXT _fl-loop.
    ELSE DO:
    /* ABL doesn't let you modify a trigger record, so delete and recreate. */
      DELETE _file-trig.
      CREATE _file-trig.
      ASSIGN _file-trig._file-recid = RECID(_file)
             _file-trig._event      = event
             _file-trig._override   = TRUE
             _file-trig._proc-name  = proc-name
             _file-trig._trig-crc   = iCRC
             ix                     = ix + 1.
    END.
  END.
END.
INPUT CLOSE.
MESSAGE ix "_file-trig records updated.".

INPUT CLOSE.
MESSAGE ix "_file-trig records updated.".
INPUT FROM "_field-t.dat". /* Field trigger data */
_fld-loop:
REPEAT:
  SET table-name field-name event proc-name.
  RCODE-INFO:FILE-NAME = proc-name.
  iCRC = RCODE-INFO:CRC-VALUE.
  FIND _file WHERE _file._file-name = table-name.
  FIND _field WHERE _field._file-recid = RECID(_file) AND
                    _field._field-name = field-name.
  FIND _field-trig WHERE _field-trig._file-recid = RECID(_file) AND
                         _field-trig._field-recid = RECID(_field) AND
                         _event = event NO-ERROR.
  IF AVAILABLE _field-trig
    AND _field-trig._proc-name = proc-name
    AND _field-trig._trig-crc = iCRC THEN NEXT _fld-loop.
  ELSE DO:
    DELETE _field-trig.
    CREATE _field-trig.
    ASSIGN
      _field-trig._file-recid  = RECID(_file)
      _field-trig._field-recid = RECID(_field)
      _field-trig._event       = event
      _field-trig._override    = TRUE
      _field-trig._proc-name   = proc-name
      _field-trig._trig-crc    = iCRC
      ix                       = ix + 1.
  END.
END.
INPUT CLOSE.
MESSAGE ix "_field-trig records updated.".
```

Note: As this example shows, you must use the `RECID` function rather than the `ROWID` function to reference metaschema table record locations.

Example: Determining which files are affected by a database change

The following procedure demonstrates how you can use the RCODE-INFO system handle with the TABLE-LIST and TABLE-CRC-LIST attributes to help determine which files you need to recompile based upon a database change. TABLE-LIST generates a list of all the database tables that are referenced in the procedure file. TABLE-CRC-LIST generates a list of the corresponding **table** CRC values stored in the **compiled** procedure file. This example procedure then compares the CRC values in the procedure file with those stored in the database. If any of the values differ, you get a message stating that the procedure needs to be recompiled, as shown:

```
DEFINE VARIABLE crclst AS CHARACTER NO-UNDO.
DEFINE VARAIBLE hBuff  AS HANDLE    NO-UNDO.
DEFINE VARIABLE ix     AS INTEGER   NO-UNDO.
DEFINE VARIABLE tbllst AS CHARACTER NO-UNDO.
ASSIGN
  RCODE-INFO:FILE-NAME = "main.r"
  tbllst               = RCODE-INFO:TABLE-LIST
  crclst               = RCODE-INFO:TABLE-CRC-LIST.
REPEAT ix = 1 TO NUM-ENTRIES(tbllst)
  CREATE BUFFER hBuff FOR TABLE ENTRY(ix, tbllst).
  IF hBuff:CRC-VALUE = INTEGER(ENTRY(ix, crclst)) THEN
    MESSAGE "main.r does not need to be recompiled".
  ELSE
    MESSAGE "main.r needs to be recompiled".
  DELETE OBJECT hBuff.
END.
```

This procedure does not generate any warnings or error messages. If the r-code file was not compiled with any tables, then the attribute returns the empty string. If the file is not found or is corrupted, the attribute returns the Unknown value (?).

Example: Determining which files differ between application versions

The signature value is a unique key that is stored within a compiled procedure. Changing one character in a procedure file results in a different signature value. The following example demonstrates how you can use the RCODE-INFO system handle with the SIGNATURE-VALUE attribute to help determine which r-code files need to be replaced in your application between versions.

```
DEFINE VARIABLE signatureVal AS CHARACTER NO-UNDO.

RCODE-INFO:FILE-NAME = "main.r".
signatureVal = RCODE-INFO:SIGNATURE-VALUE.
MESSAGE "Signature value of main.r: " signatureVal.
```

This code results in the following output:

```
Signature value of main.r: zaaJAOedxIiFQfhOSZK+lcYbOEia/oF2YUOLQPCOiUM=
```

You could create a program that runs this code for two versions of the same procedure file and then compare the resulting output. If they are the same, you know the contents of the procedure file has not changed between the two versions.

C

OpenEdge Application Limits

The hardware and operating system version on which applications run often determines the limits for applications. This appendix lists the limits you must consider when developing an application.

For information about database limits, see *Manage the OpenEdge Database*.

For details, see the following topics:

- Input/output limits
- Sorting limits
- Name limits
- Compiler limits

Input/output limits

The following table lists input and output limitations for terminals, printers, files, and streams.

Table 33: Input/output limits

Device	Operating system	Limit
Terminal line width	UNIX	80 to 132 columns
	Windows	80 columns

Device	Operating system	Limit
Printer line width	All	1 to 255 columns
Export file	All	1 to 3,000 characters per field; 32K bytes per record
Import file	All	1 to 3,000 characters per field; 32K bytes per record
Input file	All	At least 2GB, and up to 2^{63} bytes if the operating system allows
Output file	All	At least 2GB, and up to 2^{63} bytes if the operating system allows
Streams	All	1 to 50 streams per procedure

Note: Windows batch files limit you to no more than nine input parameters. This is a Windows limitation; actual executables handle more than nine.

Sorting limits

The following table lists the limits for sorting data (unless otherwise specified).

Table 34: Sorting limits

Category	Limit
Levels	1 to 16 columns or expressions.
Size	Typically, 1 to 197 bytes (that is, approximately 200 bytes minus some overhead per column and expression). The size limit depends on the number of columns and expressions you are sorting. Each column and expression uses up some number of available bytes.

Name limits

The following table lists the maximum number of characters you can use in names (unless otherwise specified).

Table 35: Name limits

Name type	Operating system	Limit
External procedure	UNIX	1 to 58 characters, plus a 2 character extension.
	Windows	1 to 255 characters, plus a 2 character extension.

Name type	Operating system	Limit
Pathnames	All	1 to 255 characters.
Table Field Field-level widget Index	All	1 to 32 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % & Names must begin with a letter.
Local variable of a routine Variable data member of a class Property of a class Run-time routine parameter	All	1 to 128 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % & Names must begin with a letter.
Internal procedure Block label Query Frame widget Browse widget Menu widget Submenu widget Menu item widget	All	No set limit; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % & Names must begin with a letter.
Stream	All	1 to 12 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % & Names must begin with a letter.
Tables in the convmap file: Attribute table Collation table Conversion table Case table	All	1 to 20 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and the hyphen character (-).Names must begin with a letter.

Compiler limits

The following table lists the limits for each element of the Compiler (unless otherwise specified).

Table 36: Compiler limits

Element	Limit
Variables	32,000 bytes for `UNDO` and `NO-UNDO` variables per external procedure. 32,000 bytes for local `UNDO` and local `NO-UNDO` variables per internal procedure or trigger block.
Statements	1 to 2147483647characters per statement. Use the Input Characters (`-inp`) startup parameter to limit the number of characters allowed in a single statement. The number of tokens allowed per statement is limited only by the available system resources. Each word or special character, such as a parenthesis, aplus sign, and a minus sign, counts as one token. Use the Token (`-tok`) startup parameter to limit the number of tokens allowed in a single statement.

[4] A routine includes a method of a class, a property accessor, a procedure, or a user-defined function.

Element	Limit
Frames	1 to 320 characters.[5]
Nested blocks	20 to 20000 blocks, including called procedures. Use the Nested Blocks (`-nb`) startup parameter to limit the number of nested blocks allowed in a procedure.

For more information about the Input Characters (`-inp`), Token (`-tok`), and Nested Blocks (`-nb`) startup parameters, see *Startup Command and Parameter Reference*.

[5] The run-time limit is 131,072.

D

Deployment Utilities and Scripts

This appendix provides reference information for the OpenEdge® deployment utilities and scripts.

For details, see the following topics:

- Templates
- Utilities summary
- DBRSTRCT utility
- MKDUMP utility
- XCODE utility
- _tlr Module

Templates

You can use the templates provided with OpenEdge, tailor them to the needs of your application, and supply them with your application. The following templates are located in the *install-dir*/toolkit/samples directory:

- **runmenu** — Starts a single-user OpenEdge session
- **multmenu** — Starts a multi-user OpenEdge session
- **runbatch** — Starts a single-user batch OpenEdge session
- **multbat** — Starts a multi-user batch OpenEdge session

- **newdb** — Makes a copy of a basic application database
- **strtsrvr** — Starts a multi-user OpenEdge server
- **stopsrvr** (supplied on UNIX only) — Stops a multi-user OpenEdge server
- **deldb** — Deletes a database
- **dumpdb** — Dumps an application database
- **reloaddb** — Loads an application database
- **trimlog** — Discards old information from the database log file
- **upgrade** — Upgrades an existing application with new data definitions and/or procedures

Utilities summary

The following table provides a summary of the different utilities provided with OpenEdge.

Table 37: Deployment utilities

Utility	Description
DBRSTRCT	Defines the type of access that all users, including you, have to your application
MKDUMP	Creates procedures a user runs to dump and reload your database
XCODE	Encrypts source code for shipment to customer sites
_tlr	Lets you tailor the environment variables used on UNIX scripts

DBRSTRCT utility

Defines the access that all users, including you, have to your application database.

Syntax

```
dbrstrct database-name
```

database-name

The name of the database for which you want to restrict access.

Notes

- Before you define database access, it is very important that you make a copy of your application database and precompile all of your application procedures against that database. For example, suppose you do not

precompile a procedure that updates the database and then you deny update access to the database. You will be unable to compile that procedure against the newly restricted database.

- Keeping a copy of the unrestricted database also lets you redefine the database restrictions at a later time. To redefine database restrictions, dump the data from the restricted database, make a copy of the unrestricted database, define restrictions on that database copy, and reload the data into it.

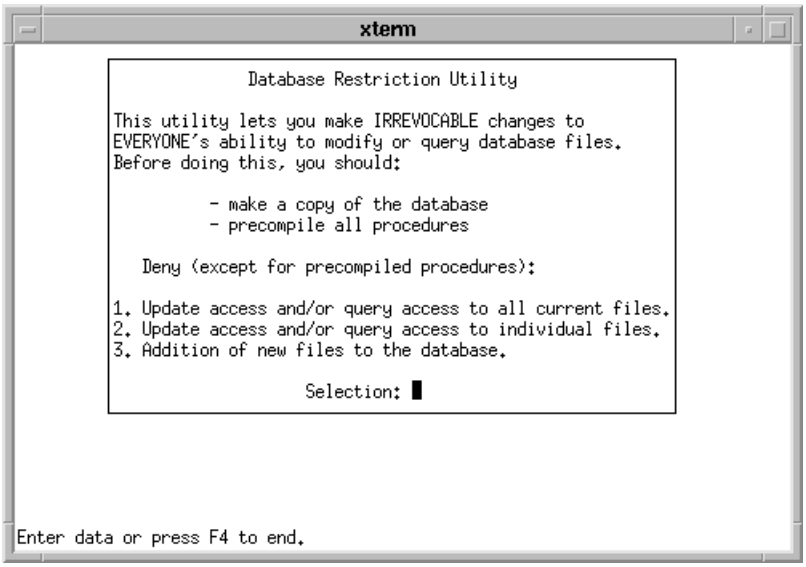
Example

Sample `DBSTRCT` command:

```
dbrstrct mydb
```

As you can see from the options listed in the following image, you can define access globally, for all files, or you can define access on a file-by-file basis. Any restrictions you define affect all users, including yourself.

Figure 21: Database Restriction Utility



If you choose option 1 and deny update access but permit query access, no users, including yourself, will have update access to files or fields in the database, unless that update is being done by a procedure that was precompiled against the database before the database was restricted. However, all users will be able to write procedures that query the database.

MKDUMP utility

Creates a set of file dump and file load procedures for a database.

Syntax

```
mkdumpc database-name
```

database-name

The name of the database for which you want to create dump and load procedures.

Note

The `filedump.p` and `fileload.p` procedures created by `mkdump` are meant to run from an application menu inside OpenEdge. If you plan to have end users run these procedures directly from the operating system level, you must first modify `filedump` and `fileload`. Comments in the `dumpdb` and `reloaddb` scripts or batch files explain these modifications.

Example

Sample `MKDUMP` command:

```
mkdump mydb
```

This command creates two main ABL (Advanced Business Language) procedures: `filedump.p` and `fileload.p`. In addition, it creates two subdirectories: `dmpprocs` and `ldprocs`. The `dmpprocs` subdirectory contains a single dump procedure for each file in the database; the `ldprocs` subdirectory contains a single load procedure for each file in the database.

An end user runs the `filedump.p` procedure to dump the database. That procedure runs the procedures in the `dmpprocs` subdirectory. To load the database, the user runs `fileload.p`, which runs the procedures in the `ldprocs` subdirectory.

XCODE utility

Encrypts ABL procedures and include files using either the default or a specified encryption key.

Syntax

```
xcode [ -k key] [ -d directory] { files | - }
```

key

A character string up to eight characters long. On UNIX, *key* is case sensitive. If you do not specify *key*, XCODE uses a default key. When OpenEdge is started with the Encrypted Compiler Mode (`-rx`) startup parameter, ABL automatically uses this default key, unless you specify the `XCODE` option on the `COMILE` statement or set the `XCODE-SESSION-KEY` attribute on the `SESSION-POLICY` handle. (The `XCODE` option takes precedence over the `XCODE-SESSION-KEY` attribute if both are in effect for a compile operation.)

The XCODE utility uses the default codepage of the operating system where it runs. To avoid possible conflicts from automatic codepage conversions, key values should use only US-ASCII characters. These characters are included in all codepages and can be located below code point 128 in the codepage.

Note that keys longer than eight characters do not generate warning or compile errors. Both the XCODE utility and the `COMPILE` statement ignore extra characters if present.

directory

The directory where the encrypted *files* are placed. The specified *directory* must be different from the source directory, because each encrypted file retains the name of its original source file. Existing files in *directory* are overwritten only if they are encrypted.

files

The pathnames, relative to the root source directory, of files to be encrypted. The pathnames supplied are appended to *directory*. Therefore, you should build a directory structure exactly parallel to the source directory structure, move to the source root, and specify relative pathnames.

- SYS$INPUT

Indicates that source filenames are to be read from standard input.

Example

```
cd /applr001
xcode -k key1 -d applx001 *.p
```

```
find applf002/*.p | xcode -k key2 -d applx002 -
```

Notes

- You must create the full directory structure desired for your encrypted files before running XCODE; XCODE does not create any directories implied by *directory* or *files*.
- The encryption algorithm is based on known techniques; therefore, the code is breakable. If your source code is highly sensitive, consider additional security methods.
- Encrypted include files must use the same *key* as any procedure that includes them.
- Compile encrypted procedures with the XCODE option on the COMPILE statement. (The XCODE option is not required if the procedures were encrypted with the default key or if the key is provided at the session-level by the XCODE-SESSION-KEY attribute of the SECURITY-POLICY handle.) Whatever the source, the key used by the COMPILE statement must match the key used with the XCODE utility. *OpenEdge Development: ABL Reference* describes the COMPILE statement in detail.

_tlr Module

Changes the default values of environment variables as defined in various scripts.

Syntax

```
_tlr var-namereplace-valuefile-name. . .
```

```
_tlr var-namereplace-value -< file-list
```

The first syntax format processes one file at a time. The second processes a list of files read from a file.

var-name

A constant character string that identifies the environment variable whose value you want to modify.

replace-value

A constant character string that identifies the new default value of the environment variable.

file-name

The name of the file in which you want to modify the default value of the environment variable.

file-list

The name of a file containing a list of filenames in which you want to modify the value of the environment variable.

Example

Several of the scripts provided with OpenEdge name the `DLC` environment variable, as follows:

```
DLC=${DLC-/usr/dlc}
```

In this example, if the value of the `DLC` environment variable has been defined, the script uses the value of that variable. Otherwise, it uses `/usr/dlc` as the value of `DLC`.

To change the default value of the `DLC` variable to `/tmp/dlc` in a script named `test.dat`, use this command:

```
_tlr DLC /tmp/dlc test.dat
```

This command changes the environment variable line in `test.dat` from:

```
DLC=${DLC-/usr/dlc}
```

To:

```
DLC=${DLC-/tmp/dlc}
```

Index

-mmax startup parameter
  execution buffer ceiling *195*
.rcd files *113*
  See also R-code swap (.rcd) files

64-bit versus 32-bit r-code *202*

A

Action code segment
  r-code *190*
Application databases
  modifying *163*
Application deployment
  CRC validation *204*
Application files
  distributing *112*
Applications
  accessing databases *25*
  administration *22*
  deployment *21*
  limits *213*
  maintaining user environments
    in Windows *51*
    on UNIX *82*
  performance *105*
Authorization
  application *39*
  compile-time *39*
  connection *39*
  databases *39*
  run-time *39*
Auto-connect feature *33*

B

Blank user IDs *45*
buildenv scripts
  and the OEBuild utility *182*
  maintaining *83*
  sample *83*
BUNDLE utility *172*, *174*
ButtonImageBorderMode option *58*, *80*
Buttons
  borders *58*

C

client.mon files *108*, *198*
Colors, See progress.ini file, PROTERMCAP file
Compile-time authorization *39*
Compile-time security
  establishing *46*
Compiler limits *215*
CONNECT statement
  handling connection denials *34*
  precedence over auto-connect *33*
Connecting
  at startup *28*
  connection modes *26*
  connection parameters *27*
  connection techniques *27*
  from the Data Administration tool *31*
  from the Data Dictionary *29*
  handling connection denials *34*
  non-Progress databases *33*
  over wide area networks *34*
  reducing connection times *34*
  specifying logical database names *26*
  with auto-connect *33*
  with the CONNECT statement *29*
Connection authorization *39*
Connection parameters, See Startup parameters
CRC, See Cyclic redundancy check
CRC calculation
  database
    metaschema fields *204*
    tables *205*
  r-code *207*
CRC compared to time stamps
  r-code availability *206*
CRC validation *203*
CRC-based r-code
  upgrading applications *163*
Cyclic redundancy check
  and r-code *46*
  database
    advantages *206*
    calculation *204*
    changes affecting *205*
    code validation *203*
    deployment *206*
    disadvantages *207*
  r-code
    application procedures *208*
    calculation *207*
    RCODE-INFO handle *208*
    schema triggers *208*

D

-D startup parameter, Directory Size *195*, *198*

Data Administration tool
connecting databases *31*
defining compile-time security *42*
determining privileges *45*
disconnecting databases *37*
selecting a working database *36*
Data Dictionary
connecting databases *29*
defining compile-time security *42*
determining privileges *45*
disconnecting databases *36*
performing security administration *49*
selecting a working database *35*
temporary changes (.trp) file *113*
Database CRCs
database changes affecting *205*
Databases
authorization *39*
connecting *25*
CRC
code validation *203*
disconnecting *36*
logical names *26*
non-Progress *33*
restricting access *179*
schemas *34*
selecting a working database *35–36*
startup without connecting *35*
temporary files *113*
time stamps *203*
DataServers *163*
.dbi files *113*
DBRSTRCT utility *148*, *157*, *179*, *218*
Debugger segment
r-code *190*
Default window characteristics, See progress.ini file
Defining debugger buttons, See progress.ini file
Defining debugger macros, See progress.ini file
deldb template *217*
Deploying applications
steps *151*
Deployment
See also Applications
process *21*
requirements *21*
services *21*
tasks *22*
See also Applications
Direct printing *133*
Directory Size (-D) startup parameter *195*, *198*
Disconnecting databases
with the Data Administration Tool *37*
with the Data Dictionary *36*
Dual-image buttons *58*
dumpdb template *217*
Dumping *177–178*

E

Edit buffer contents files *113*
Encrypted source code
steps for deployment *154*
upgrading applications *163*
Encrypting procedures *169*, *171*
Encryption keys *147*
Environment characteristics, See progress.ini file
Environment variables
specifying *78*
UNIX *182*
Windows *185*
Execution buffer
available memory *195*
ceiling
described *196*
size *195*
described *194*, *196*
Execution environment
limits *195*, *198*
r-code
typical layout *194*, *196*
statistics
.srt file access *198*
execution map *198*
segment information *198*
startup parameters *198*
summary information *198*
Expression code segment
r-code *190*
Extended character support
with progress.ini file entries *76*
with PROTERMCAP file entries *101*

F

FastCursorMode option *70*
File descriptor limits *25*
File system recovery program (fsck) *113*
Files
buildenv scripts *182*
client.mon *108*
distributing files on a network *111*
edit buffer contents files *113*
local before-image files *113*
parameter files *27*
printer control files *139*
proc.mon *108*
procedure library (.pl) files *118*
progress.ini file *56*, *112*
PROTERMCAP file *83*
sort (.srt) *113*
temporary changes files *113*
temporary files *113*
temporary table files *113*

Fonts, See progress.ini file
Frame segment
r-code *190*
fsck (file system recovery program), See File system recovery program (fsck)
Full development products
list of *152*

I

Image buttons
Windows XP look *58*
IMAGE environment variable *183*
Index modifications *148*
INI2REG utility
See also Registry
command line interface *53*
See also Registry
Initial value segment
r-code *190–191*
Input/output device limits
number of characters *213*
Internationalization *146*, *148*

K

KERMIT *153*
Key bindings, See progress.ini file, PROTERMCAP file
Keymapping
See also progress.ini file
Version 6 *93*
See also progress.ini file

L

.lbi files *113*
LIBRARY function *120*
licensing *182*
Limits
compiler *215*
input/output devices
number of characters *213*
name limits
number of characters *214*
sorting *214*
Loading *177–178*
Local before-image files *113*
Logical database names
changing with the ALIAS statement *26*

M

Manifest file *51*, *58*, *80*
Maximum Memory (-mmax) startup parameter *195*
MEMBER function *120*
Memory-mapped procedure library, See Procedure libraries
Metaschema tables
CRC calculations *204*
MKDUMP utility *177*, *219*
multbat template *217*
multmenu template *217*

N

Name limits
number of characters *214*
newdb template *217*
Non-English character support *101*
Non-Progress databases *33*

O

OEBuild utility *181*
OpenEdge GUI for .NET applications
steps for deployment *159–160*
Operating system security *47*
Output routing *136*
OUTPUT THROUGH statement *135*
OUTPUT TO statement
PRINTER option *134*

P

Parameter files, See Startup parameters
.ped files *113*
Performance
distributing files on a network *111*
monitoring r-code activity *108*
procedure loading and execution *105*
reducing database connection times *34*
sorting *114*
temporary file I/O *113*
using procedure libraries *106*
Permissions
See also Security
blank user ID *45*
CAN-CREATE *42*
CAN-DELETE *42*
CAN-DUMP *42*
CAN-LOAD *42*
CAN-READ *42*
CAN-WRITE *42*
table *47*
See also Security
Platforms
r-code *202*
Portability
encrypted source *147*
r-code *148–149*

Portability *(continued)*
unencrypted source *146*
Portable r-code *201*
Print queues *133*
Printers
administration *133*
configuration *139*
control files *139*
control sequences *139*
defaults *134*
Printing
direct *133*
in Progress *133*
in Windows *135*
output routing schemes *136*
OUTPUT THROUGH statement *135*
OUTPUT TO statement *134*
pagination *134*
PUT CONTROL statement *139*
spooled *133*
proc.mon files *108*
Procedure libraries
See also PROLIB utility
advantages *106*
and code pages *131*
and PROPATH *120*
and r-code performance *106*
listing contents *127*
memory and network considerations *121*
memory-mapped format
generating *124*
loading and executing *107*, *118*
overview *116*
PROPATH *193*
running procedures
using a relative pathname *119*
using an absolute pathname *119*
setting up *118*
standard format
adding files *125*
advantages *116*
compressing files *130*
creating *124*
deleting files *126*
extracting files *128*
loading and executing *107*, *117*
memory-mapped format
advantages *116*
replacing files *126*
tuning *107*
See also PROLIB utility
Procedures
distributing source files *112*
r-code integrity
example *208*
running from procedure libraries *119*
Product upgrades *167*
Proenv *52*
Progress system files
distributing *112*
progress.ini file
defining debugger buttons *75*
defining key bindings *73*
defining key mappings *76*
maintaining *56*
search path *54*
sections
colors section *67*
debug-buttons section *75*
debug-init section *74*
debug-macros section *75*
default window section *68*
defining debugger macros *75*
fonts section *69*
keymap section *76*
keys section *75*
startup section *58*
winchar colors section *72*
winchar default window section *73*
winchar keys section *73*
winchar startup section *70*
specifying colors *67*, *72*
specifying default window characteristics *68*, *73*
specifying environment characteristics *58*, *70*
specifying environment variables *78*
specifying fonts *69*
specifying key bindings *75*
PROLIB utility
See also Procedure libraries
parameters *122*
sample commands *130*
using *122*
using wild cards *123*
See also Procedure libraries
PROLOAD environment variable *182*, *185*
PROTERMCAP file
copying a UNIX termcap file *102*
copying an existing terminal entry *104*
defined *83*
general syntax *84*
keyboard mappings *96*
maintaining *83*
non-English character support *101*
PROTERMCAP environment variable *103*
setting the terminal type *102*
size restrictions *83*
specifying colors *93*
syntax *84*
terminal capabilities entries
for cursor motion *94*
terminal name entries *86*
Vermont Views *91*

PROUTIL utility *46*
prowin32.exe.manifest file *80*
PUT CONTROL statement *139*

R

R-code CRC
RCODE-INFO handle *207*
procedure integrity *207–208*
schema triggers *208–209*
R-code directory
management *198*
size *195*
described *194*, *196*
R-code file
segment layout *192*
R-code swap (.rcd) files *113*, *196*
R-code
See also Procedure libraries
and cyclic redundancy check (CRC) *46*
directory
defined *106*
management *198*
described *194*
size *107*
files
segment layout *191*
64-bit versus 32-bit *202*
execution buffer *106*, *194*
steps for deployment (CRC) *157*
text segment *190*
portability *201*
action code segment *190–191*
segment descriptor table *106*, *194*, *196*
frame segment *190*
application databases *166–167*
execution
environment limits *195*, *198*
sequence *195*, *197*
segment management *195–196*, *198*
available memory *196*
environment statistics *198*
segment types *190*
initial value segment *190–191*
session sort file *106*
memory-mapped execution environment *196*
performance *111*
and code pages *131*
features *190*, *193*
directory description *196*
execution buffer described *196*
factors affecting size *191*
tuning *107*
expression code segment *190*
shared memory buffer *106*
structure *190*
R-code *(continued)*
defined *105*, *189*
loading and executing
from a memory-mapped library *107*, *118*
from a standard library *107*, *117*
from a file *107*, *117*
monitoring activity *108*
CRC *207*
standard execution environment *194*
execution environment
typical layout *194*, *196*
debugger segment *190*
See also Procedure libraries
R-code segments *190*
RCODE-INFO handle *208*
Registry
See also INI2REG Utility
updating *51*
See also INI2REG Utility
reloaddb template *217*
Run-time authorization *39*
Run-time security
establishing permissions tables *47*
runbatch template *217*
runmenu template *217*

S

Schema cache files *34*
Schema holder *33*
Schema triggers
auto-deployment example *209*
r-code CRC
deployment *209*
RCODE-INFO handle *208*
setting *208*
validating *208*
R-code CRC *208*
SEARCH function *120*
Security
administration *48*
blank user ID *45*
compile-time *46*
field-level *46*
operating system *47*
run-time *47*
table-level *46*
SECURITY-POLICY
SET-CLIENT() method *41*
Segment descriptor table *194*, *196*
Segment layout
r-code file *192*
Segment management
r-code *196*, *198*
Segment Statistics (-yd) parameter *198*

Segmented r-code
types 190
Session files
distributing 112
SET-DB-CLIENT() ABL function 41
SETUSERID() ABL function 41
Single-image buttons 58
Sort (.srt) files 113
Sorting
improving performance 114
limits 214
Source files
distributing 112
Spooled printing 133
Standard library, See Procedure libraries
Startup parameters
Database Name (-db) 26
Directory Size (-D) 107
Host Name (-H) 26
Input Characters (-inp) 215
Internal Code Page (-cpinternal) 124, 131
Logical Database Name (-ld) 26
Maximum Memory (-mmax) 107, 110
Merge Number (-TM) 114
Nested Blocks (-nb) 107, 215
Number of Users (-n) 34
Parameter File (-pf) 27
Password (-P) 45
Printer (-o) 134, 136
PROLIB Memory (-plm) 108, 121
PROLIB Swap (-pls) 107–108, 117, 121
Quick Request (-q) 120
R-code In Code Page (-cprcodein) 131
Save Temp Files (-t) 113
Schema Cache File (-cache) 34
Segment Statistics (-yd) 108, 111
Service Name (-S) 26
Single-user Mode (-1) 26
Speed Sort (-TB) 114
Stack Size (-s) 107
Statistics (-y) 108, 110
Statistics with Cross-reference (-yx) 108
Statistics with CTRL-C (-yc) 108
Temporary Directory (-T) 112–113
Token (-tok) 215
User ID (-U) 45
Windows Passthrough Printing (-wpp) 135
Statistics (-y) startup parameter 198
stopsrvr template 217
Storage media
space used 146
strtsrvr template 217

T

Table permissions
CAN-CREATE 42
CAN-DELETE 42
CAN-DUMP 42
CAN-LOAD 42
CAN-READ 42
CAN-WRITE 42
Tables
CRC calculation 205
Templates 217
Temporary changes files 113
Temporary files
defined 113
distributing 112
Temporary table files 113
TERMINAL function 102
Terminals
See also PROTERMCAP file
UNIX stty control functions 100
See also PROTERMCAP file
Text segments
r-code 190
translation 191
_tlr module 221
trimlog template 217
.trp files 113

U

Unencrypted source
steps for deployment 153
upgrading applications 163
Unicode 58
UNIX environment variable 182
Upgrade templates 165, 217
Upgrades
Progress products 167
UseClipChildren option 58, 80
UseNative3D option 80
User count
exceeding 34
User environments
maintaining in Windows 51
maintaining on UNIX 82
Utilities 25
See also Data Administration tool

W

Web applications
deploying 201
WebSpeed 70
Wide area networks (WAN)
connecting 34

Widgets
XP look *58*
Wild cards
and the PROLIB utility *123*
Windows icons *79*
Windows XP look *58*, *80*
Working databases
selecting *35*

X

XCODE utility *169*, *171*, *220*
XP look *58*, *80*

Y

-y startup parameter, statistics *198*
-yd startup parameter, Segment Statistics *198*